Proyecto Fin de Grado

Grado en Ingeniería Informática

Aplicación web para la gestión de una bolsa de horas

Autor: Alberto De Ávila Hernández
Tutor: Telmo Zarraonandia Ayo

Leganés, 7 de Junio de 2012

Título: Aplicación web para la gestión de una bolsa de horas.

Autor: Alberto De Ávila Hernández
Director: Telmo Zarraonandia Ayo

EL TRIBUNAL

Presidente: Ricardo Colomo Palacios

Vocal: Raúl Arrabales Moreno

Secretario: Manuel Carretero Cerrajero

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 3 de Julio de 2012 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

A mis padres, Liborio y Esperanza, pos enseñarme a ser ambicioso y a querer llegar más allá de las metas que me marqué, por fomentar mi intelecto y enseñarme que, con esfuerzo y persistencia, los sueños se cumplen.

A mi hermano Diego, por enseñarme todo lo que yo no podía imaginar y mostrarme el camino.

A Alba, por aguantar todo el tiempo dedicado, no solo al proyecto, sino también a la carrera y por apoyarme cuando más lo necesité.

A mis compañeros de carrera, Adrián, Alberto, Alejandro, Alfonso, Álvaro, Daniel, David, Javier, Manuel, Miguel, Santiago y Víctor, por compartir su conocimiento y sabiduría conmigo, porque sin ellos no habría llegado tan lejos y por tantos momentos.

A Álvaro, Fátima, Gerardo y Miguel, por haberme dado la oportunidad de formar parte de Salenda, por la paciencia que han demostrado tener conmigo y por el conocimiento que me han aportado.

A Telmo Zarraonandia, por su predisposición y las horas dedicadas a este trabajo.

Sinopsis

El presente documento describe las distintas fases asociadas al diseño y desarrollo de una herramienta que tiene por objetivo informar al usuario del estado de la bolsa de horas de soporte que tiene contratada una determinada empresa. La funcionalidad principal de la aplicación es permitir al usuario conocer cuántas horas de soporte le restan, cuántas han sido consumidas y en qué tareas han sido empleadas, así como obtener distintos tipos de estadísticas con los datos asociados a esta información.

La particularidad de esta aplicación es que los datos que utiliza, salvo la información de los usuarios, es obtenida de la interacción con otras dos herramientas externas, que son JIRA, un gestor de tareas o incidencias, y Factura Directa, una herramienta para gestionar clientes y facturas. Ambas herramientas son propietarias y el acceso a la información que contienen se realiza mediante un API que utiliza la tecnología REST.

Además de REST, la aplicación será desarrollada utilizando como lenguaje de programación Groovy, un lenguaje orientado a objetos que se ejecuta sobre la plataforma Java y Grails, un *framework* para el desarrollo con Groovy. Cabe destacar también que se va a emplear para el desarrollo del proyecto una metodología ágil, Scrum, que se basa en el trabajo iterativo e incremental.

La aplicación cumplirá los requisitos establecidos por la empresa de desarrollo del software Salenda, que en este caso personificará el rol de cliente.

Abstract

This document describes the different stages associated with the design and development of an application that aims to inform the user about the status of the stock of support hours that a company has contracted. The main funcionality of the application is to allow the user to know how many support hours are left, how many have been used and see statistics about the associated information.

The particularity of this application is that all the data used, except user's information, is obtained from the interaction whit another two applications, JIRA, a task manager, and Factura Directa, a tool for managing clients and invoices. Both applications are propietary and their data will be acceded using the REST API.

In addition to REST, the application will be developed using as programming lenguage Groovy, an object-oriented language that runs on Java, and Grails, a framework for development with Groovy. Also during the step of development, an agile methodology called Scrum will be used. This methodology is base on iterative and incremental work.

The application shall meet the requirements of the software development company Salenda, which in this case impersonates the client role.

Índice general

Índice de ilustraciones

Índice de tablas

Sección 1

1 Introducción y objetivos

En este apartado se detallan los objetivos que han propiciado la necesidad de desarrollar esta aplicación, se enumeran los medios técnicos que se emplearan para ello y las diferentes tecnologías. Además se detalla la estructura que va a seguir este documento.

1.1 Objetivos

Los diferentes objetivos que se pretenden conseguir con el desarrollo de este proyecto de fin de grado van, desde conseguir un mayor grado de conocimiento, no solo a la hora de desarrollar aplicaciones desde cero, sino también de las diferentes tecnologías web o el estado de aplicaciones web en la actualidad, pasando por crear una aplicación útil que permita, a los clientes de la empresa de software Salenda [1], gestionar las horas de la bolsa de soporte.

Dentro de la aplicación, los usuarios podrán consultar el número total de horas que les restan para seguir ofreciendo soporte a sus aplicaciones o desarrollando nuevas funcionalidades, así como el tiempo ya dedicado en cada una de las tareas desarrolladas, entendiendo por tarea todas las actividades que requieran tiempo para su resolución. Además, pretende aportarme un mayor grado de madurez como profesional de la informática y como ingeniero, haciéndome capaz de resolver problemas con una serie de condiciones que me hacen exigirme a mi mismo más y desempeñar todo mi intelecto.

En este documento se irá plasmando el continuo trabajo llevado a cabo para conseguir que la aplicación quede desarrollada y funcionando utilizando los recursos asignados y en el tiempo acordado, así como las diferentes fases por las que ha atravesado.

1.2 Motivación de la aplicación

La aplicación para la gestión de la bolsa de horas nace de la necesidad de ofrecer, a los clientes de Salenda, una forma de tener acceso al saldo que les resta de su bolsa de horas de soporte. Las horas de soporte se pueden gastar en desarrollar nuevas funcionalidades o corregir fallos en aplicaciones que no estén en garantía. Al cliente se le mostrará un desglose de las diferentes incidencias que han consumido horas de la bolsa de soporte. La información sobre las horas que se van gastando en cada tarea, proviene de una aplicación externa, obteniendo los datos por medios que explicaremos en sucesivos apartados. Además de esta información, también se obtiene información de los diferentes proyectos que se están desarrollando y de los clientes.

1.3 Etapas del proyecto

El desarrollo de esta aplicación se ha dividido en las siguientes fases:

1. Análisis de las necesidades para el desarrollo de la aplicación y comparación de las tecnologías en el desarrollo de aplicaciones web.
2. Definición de los objetivos a conseguir.

3. Especificación de los requisitos del sistema.
4. Planificación y presupuesto.
5. Diseño de la aplicación.
6. Para cada fase: Desarrollo, pruebas, reunión y posibles cambios en requisitos.
7. Puesta en producción.

1.4 Medios para el desarrollo del proyecto

- Una licencia de Atlassian JIRA [2].
- Una licencia de Atlassian Confluence [3].
- Una licencia de GreenHopper [4].
- Una base de datos H2 [5].
- Un servidor Glassfish [6], ejecutándose sobre una distribución Linux, Ubuntu [7].
- Un Macbook con SpringSource Tool Suite [8] instalado.

1.5 Estructura de la memoria

La memoria del proyecto está compuesta de la siguiente forma:

Introducción y objetivos.

En este apartado se van a detallar el preámbulo de la memoria, así como los principales objetivos que se quieren conseguir con su desarrollo, incluyendo la estructura de la memoria.

Tecnologías y herramientas.

En este capítulo se va a comentar la situación actual de las tecnologías en los desarrollos web, los diferentes lenguajes que se pueden utilizar, y los que finalmente se usarán. Además se explican las diferentes herramientas que se van a utilizar durante todo el desarrollo.

Análisis del sistema.

En esta sección se van a analizar los diferentes requisitos que presenta la aplicación, extraídos de las diferentes reuniones mantenidas con el cliente, incluyendo algunos diagramas de casos de uso.

Gestión del proyecto.

En este apartado se va a detallar la forma de administrar y planificar las tareas, detallando la metodología utilizada y la gestión del tiempo. Además se introduce una comparativa entre la estimación inicial de tiempo y lo que finalmente se ha empleado en el transcurso del proyecto.

Se incluye un cálculo del presupuesto para el desarrollo de esta aplicación.

Diseño de la aplicación.

En este capítulo se especifica la arquitectura que va a seguir la aplicación desarrollada, el modelo de datos de la aplicación a desarrollar y de las aplicaciones de las que se extrae la información, las diferentes interfaces de usuario y los controladores de la aplicación.

Implementación de la aplicación.

En esta etapa se van a explicar aquellas secciones de código que, de forma ilustrativa, aporten una visión completa de cómo se ha desarrollado la aplicación.

Pruebas.

Se van a detallar algunas de las diferentes pruebas utilizadas para verificar que la funcionalidad desarrollada es correcta.

Conclusiones.

Se analizaran los resultados obtenidos del desarrollo del proyecto, los objetivos cumplidos, los conocimientos adquiridos y los aspectos a tener en cuenta en el futuro.

Anexos.

Contiene los anexos a este documento, entre los que encontramos el manual de usuario de la aplicación.

Bibliografía y referencias.

Fuentes de información utilizadas, incluyendo libros, recursos electrónicos o artículos, durante el desarrollo del proyecto y su documentación.

Sección 2

2 Tecnologías y herramientas

En este apartado se describen las diferentes tecnologías que se van a emplear para durante el desarrollo de la aplicación, las herramientas que se van a utilizar y las metodologías de desarrollo.

2.1 Desarrollo del software mediante metodologías ágiles

Las metodologías ágiles surgen de la necesidad de crear mecanismos que permitan, a los programadores, ser más rápidos a la hora de crear software y permitir responder dinámicamente a los posibles cambios durante el desarrollo. Existen diferentes metodologías como puede ser Extrem Programing (XP) o Scrum, y otras [9], todas ellas están basadas en el manifiesto ágil[10]. El manifiesto ágil es un documento redactado en 2001, que establece una serie de principios básicos que se deben seguir:

- Entregas parciales periódicas que permitan al cliente ver la evolución del proyecto.
- Se permite el cambio de requisitos.
- Se mantienen reuniones breves cada día en las que cada miembro del equipo de desarrollo expone que hizo la jornada anterior, que va a hacer esta y que problemas ha tenido.
- Los miembros del equipo se organizan solos para desempeñar sus labores.
- Deben trabajar, de forma coordinada, tanto los desarrolladores como los clientes y los usuarios.
- El software que funciona es la principal forma de medir el avance del proyecto.
- Por cada entrega, el equipo analiza los principales problemas encontrados, con el fin de intentar evitarlos en futuras entregas.

Para el desarrollo de este proyecto se utilizará Scrum [11], con el que se realizaran entregas cada 2-4 semanas (a las que llamaremos *sprints*), y se le asignan roles diferentes a cada parte interesada dentro del desarrollo de la aplicación. Estos roles son:

- **Product Owner:** es el cliente, encargado de facilitar los requisitos (denominados en Scrum historias de usuario), priorizarlos y verificar que están correctamente desarrollados.
- **Scrum Master:** persona encargada de hacer cumplir las reglas que establece Scrum y eliminar los obstáculos que impidan el avance del desarrollo.
- **Equipo de desarrollo:** encargados de implementar las historias de usuario.

En este caso, los roles de Scrum Master y de Product Owner serán llevados a cabo por los desarrolladores de Salenda y el equipo de desarrollo estará compuesto por mí.

2.2 Lenguajes de programación

Los lenguajes de programación son un conjunto de instrucciones consecutivas que se ejecutan en un procesador. Las instrucciones que ejecuta el procesador están escritas en lenguaje maquina, es decir, en lenguaje binario, incomprensible para el ser humano, por lo que se crearon lenguajes comprensibles. Se comenzó con la creación del lenguaje ensamblador, muy parecido al lenguaje maquina pero entendible. Este lenguaje tiene un gran problema y es que depende del procesador, ya que cada procesador puede tener su propio lenguaje maquina.

Los lenguajes de programación se pueden clasificar en:

- **Compilados**: lenguaje que para ejecutarse debe producirse un archivo ejecutable, a través de un compilador. El típico ejemplo de lenguaje compilado es C.
- **Interpretados**: lenguaje que debe traducirse al lenguaje maquina, a través de un programa auxiliar o interprete, para poder ser ejecutado. Un ejemplo de lenguaje interpretado puede ser PHP.

Existen otros lenguajes que son tanto interpretados como compilados, como es el caso de Java [12].

2.2.1 Java

Java es un leguaje de programación creado por Sun Microsystems [13] en 1995. Este lenguaje es uno de los más extendidos en el desarrollo de aplicaciones, juegos y herramientas. Además es una plataforma informática y está presente en miles de millones de dispositivos móviles y ordenadores [14].

2.2.2 Lenguajes dinámicos

Los lenguajes dinámicos son aquellos lenguajes que permiten la modificación del código en tiempo de ejecución y que no requieren tiempo fe compilación. En la actualidad existen muchos lenguajes dinámicos, como pueden ser PHP, Python, Ruby o Groovy, pero nos centraremos en este ultimo por estar basado en Java [15].

2.2.2.1 Groovy

Groovy es un lenguaje de programación dinámico que se ejecuta sobre la máquina virtual de Java y creado por SpringSource. Este lenguaje está influenciado por otros lenguajes dinámicos como Python, Ruby o Smaltalk, pero basado principalmente en Java. Entre las principales características, destacan

el tirado dinámico, el uso de *closures* (bloques de código que se asignan a una variable y se ejecutan a posteriori), soporte para expresiones regulares y la sobrecarga de operadores.

En la siguiente imagen se puede observar un ejemplo de una *closure*. En ella se itera sobre cada elemento de la lista para calcular el sumatorio.

```
def method = {
    def list = [4, 8, 15, 16, 23, 42]
    def sum
    list.each { element ->
        sum += element
    }
}
```

Ilustración 1. Closures de Groovy

La síntesis de Groovy es sencilla y muy similar a Java, con algunas diferencias que hacen que el lenguaje sea mucho más legible. Estos cambios son:

- No es necesario poner punto y coma después de escribir una sentencia.
- No son necesarios los paréntesis.
- Los métodos de acceso a las variables, comúnmente denominados *getter* y *setter* no aparecen, pero vienen implícitos. Si es necesario añadir funcionalidad a esos métodos, basta con escribirlos con la nueva lógica.
- Los métodos que reciben parámetros pueden ser llamados con un menor número de ellos. Por ejemplo:

```
def method(x=1,y=2) {
    x + y
}

method()    //x + y = 3
method(2)   //x + y = 4
method(2,3) //x + y = 5
```

Ilustración 2. Método en Groovy con varios parámetros

2.2.3 Javascript

Javascript es un lenguaje de programación utilizado en páginas web, que se encarga de ejecutar el navegador. Javascript dispone de muchas librerías que permiten ampliar funcionalidad y aplicar nuevos estilos en las aplicaciones web [16]. Una de las librerías más utilizadas es jQuery.

jQuery es una librería Javascript que ofrece un mecanismo fácil para interactuar con los documentos HTML, modificar los documentos HTML, crear efectos gráficos, manejar eventos e interaccionar con otras páginas web mediante tecnologías Ajax. [17]

Con el uso de esta tecnología, se consiguen ahorrar muchas líneas de código para conseguir un mismo objetivo si no utilizáramos jQuery. Por ejemplo, en la siguiente ilustración, se ejecuta una función, una vez ha cargado la página y almacena el valor de un campo cuyo identificador es “username”.

```
$(document).ready(function(){
    var inputUsername = $("#username");
)};
```

Ilustración 3. jQuery

jQuery dispone de múltiples *plugins* para ampliar su funcionalidad. Entre ellos vamos a usar:

- **jEditable**: *plugin* que permite editar un campo mostrado, sin necesidad de utilizar un formulario complejo. Como se ilustra en la siguiente imagen, cuando queremos editar un campo, pulsamos sobre el, lo editamos y solo hace falta pulsar sobre el botón de “OK” para guardar el cambio.

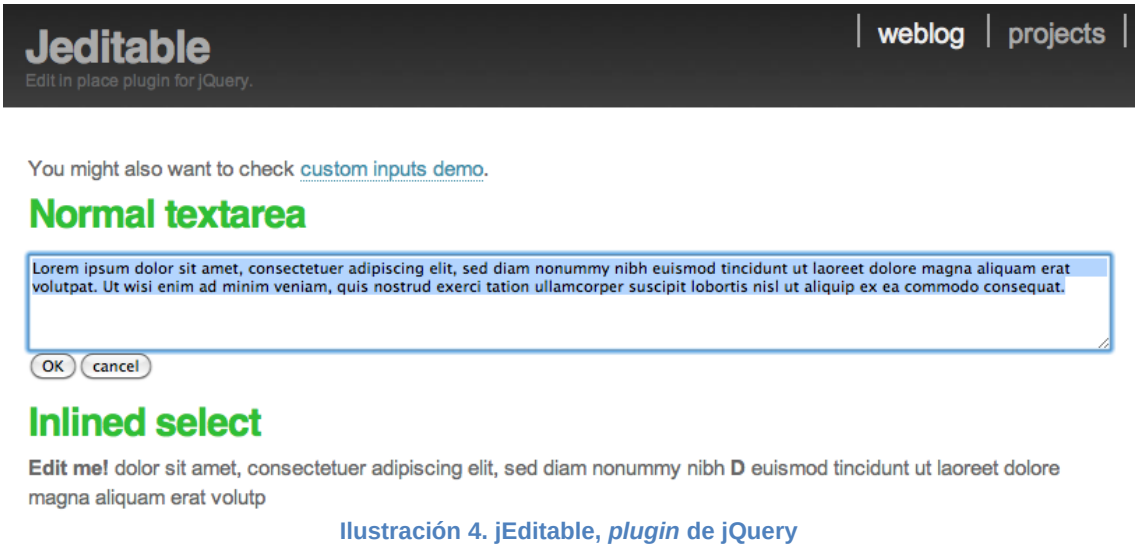

Ilustración 4. jEditable, *plugin* de jQuery

- **jQuery-ui**: es la librería oficial de jQuery para crear interfaces. Con ella podemos definir diferentes estilos para los botones, pestañas, calendarios y otros elementos visuales de HTML. [18]
  En la siguiente imagen podemos observar los diferentes estilos que se pueden aplicar a botones, pestañas, calendarios, campos de texto con autocompletado y barras de deslizamiento.

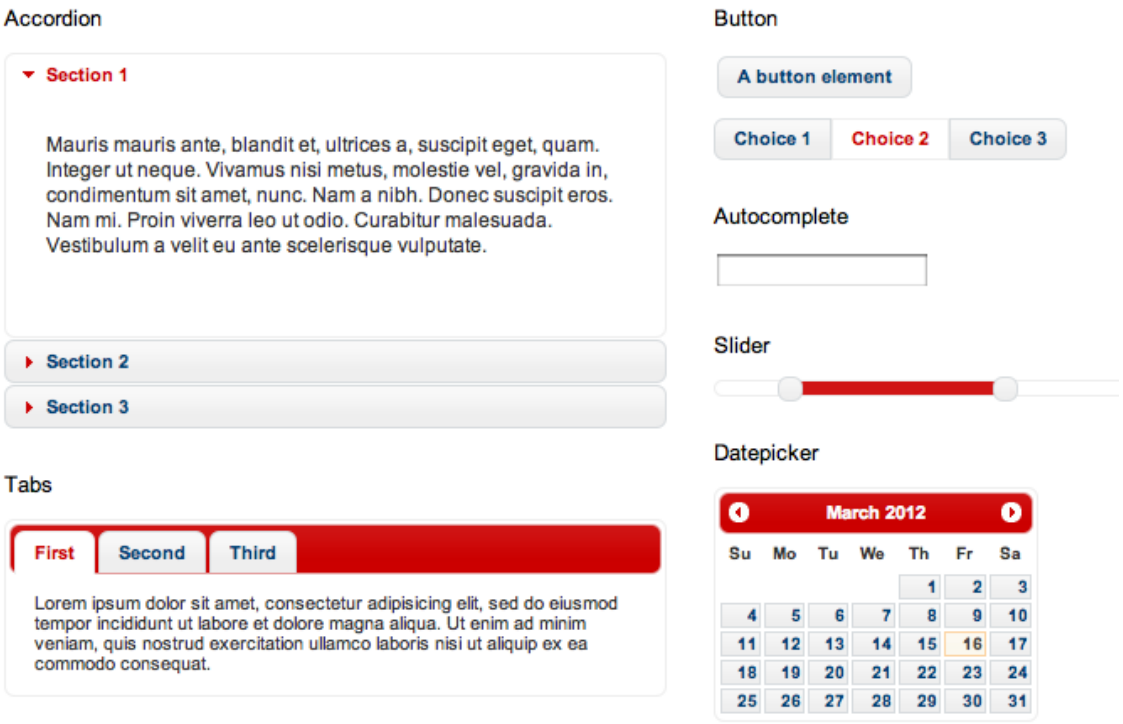

Ilustración 5. jQuery-ui

2.3 *Framework* de desarrollo

Un *framework* es una herramienta o conjunto de herramientas que permiten el desarrollo de una aplicación dentro de un determinado ámbito y no solo para el desarrollo de aplicaciones web. Podemos encontrar *frameworks* para el desarrollo de juegos o para crear aplicaciones de escritorio.

Los *frameworks* persiguen ofrecer a los desarrolladores, no solo las herramientas con las que crear la aplicación, sino también que el desarrollo sea más rápido y promover el uso de buenas prácticas de programación. [19]

Existen varios *frameworks* para el desarrollo de aplicaciones con el lenguaje Groovy. Entre ellos podemos encontrar [20]:

- **Grails.**
- **Griffon.**
- **WOGroovy.**
- **Aerie.**

2.3.1 Grails

Grails es el *framework* para el desarrollo de aplicaciones web por medio del lenguaje Groovy. Dentro de las diferentes posibilidades para la creación de aplicaciones con Groovy, Grails es de los *frameworks* más utilizados por diferentes razones [21] [22]:

- Se ejecuta sobre la máquina virtual de Java (JVM).
- Utiliza la estructura del modelo vista controlador (MVC). En esta estructura, el modelo representa los datos o información almacenada, la vista donde se muestra la información y el controlador, los mecanismos encargados de interactuar entre la vista y el modelo.
- Sigue varias filosofías:
    - *Don't Repeat Yourself* (DRY): no te repitas a ti mismo. Quiere decir que no vuelvas a escribir código ya escrito, principalmente utilizando *plugins.*
    - *Convention over Configuration*. Con Grails se intenta evitar que el programador dedique demasiado tiempo a los ficheros de configuración en XML.
- Ofrece la creación de la representación de los datos (las vistas), mediante Grails Server Pages (GSP). Las páginas GSP proporcionan las mismas funcionalidades que las Java Server Pages, y además agregan nuevas funcionalidades que permiten mostrar contenido utilizando menos líneas de código.
- Utiliza *Grails Object Relational Mapping* (GORM). GORM es la implementación Grails de un mapeo relacional de objetos (ORM). Los ORM permiten establecer una relación entre el modelo de datos y la información almacenada en la base de datos, lo que permite operar

sobre la información de manera sencilla. Además con GORM ya contamos con métodos implementados implícitamente como pueden ser *delete*, *save* o *update*.

- No es necesario implementar métodos para realizar búsquedas, porque con Grails se pueden utilizar buscadores dinámicos, es decir, si una clase de dominio “coche” tiene un atributo “modelo”, podemos buscar en la tabla coches invocando al método “coche.findByModelo(modelo)”

  La arquitectura que tiene Grails es la siguiente:

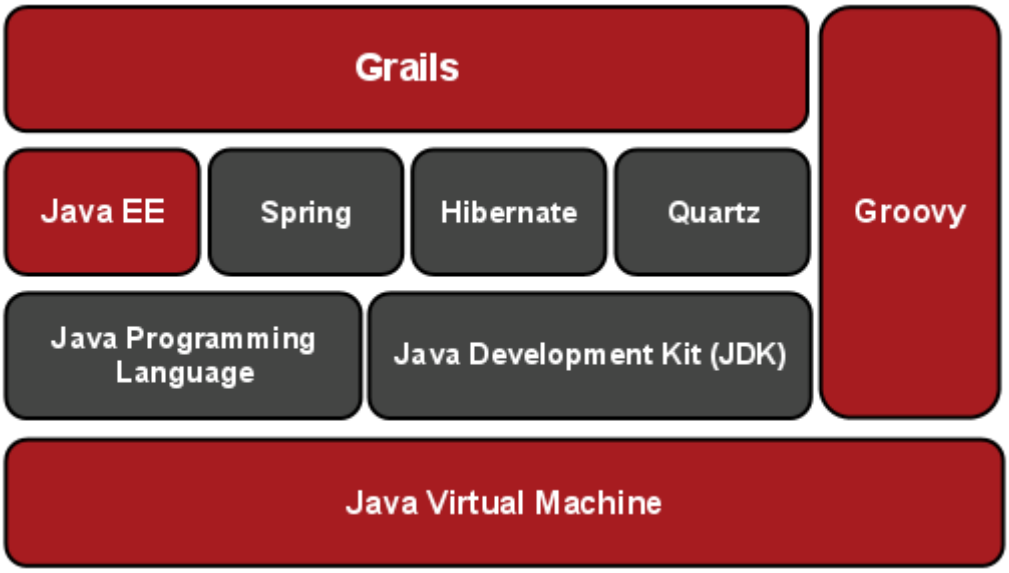

Ilustración 6. Arquitectura Grails

2.4 *Plugins*

Un *plugin* es un complemento software o hardware que permite ampliar una característica o agrega un servicio a un sistema más grande [23].

En el caso de Grails, los *plugins* son paquetes que permiten agregar funcionalidades a nuestro proyecto. Por ejemplo, existe un *plugin* que nos permite realizar búsquedas de contenido dentro de nuestra aplicación.

A continuación se detallan aquellos que se han empleado en la realización de este proyecto.

2.4.1 jQuery

Existe un *plugin* que ofrece la integración de la librería Javascript jQuery, anteriormente mencionada, con Grails. Esto nos permite tener un control sobre las extensiones que utilizamos, permitir acceder de forma más cómoda a los recursos que ofrece jQuery y conocer la versión con la que se trabaja [24].

2.4.2 jQuery-ui

Al igual que con el *plugin* de jQuery para Grails, con este conseguimos evitar duplicidades y conflicos, manteniendo de forma integrada nuestro *framework* con las diferentes extensiones que este nos permite utilizar [25].

2.4.3 Spring Security Core

Spring Security Core es un *plugin* que nos ayuda a diseñar y crear la gestión de los usuarios de la aplicación. El *plugin* nos permite definir los roles existentes dentro de la aplicación y los datos que contiene cada tipo de usuario. Además ofrece la posibilidad de restringir el acceso a los contenidos en función del rol asignado de una forma muy simple [26].

2.4.4 Quartz

Quartz es un *plugin* que permite crear tareas que se repiten cada cierto tiempo. Esto nos permite automatizar el envío de informes, consultas, actualizaciones o cualquier acción que realice nuestra aplicación. En el caso de la aplicación, este *plugin* será utilizado para obtener la información de otras aplicaciones cada noche [27].

2.4.5 Mail

Mail es uno de los *plugins* más famosos de Grails. Con el se permite el envío, de forma sencilla, desde la aplicación. Gracias a su sencillez, podemos enviar desde un simple correo con un texto, hasta adjuntar archivos al mismo. Además la configuración necesaria para su funcionamiento es básica y está documentada dentro de la documentación del *plugin* [28].

2.4.6 Settings

Settings es un *plugin* con el que se permite almacenar en base de datos aquellas variables de configuración necesarias en la aplicación y que no necesitan una tabla cada una de ellas para ser almacenadas. Además el *plugin* nos ofrece una sencilla manera de acceder a los valores, tanto desde la vista como desde los controladores, dentro de la arquitectura de la aplicación [29].

2.4.7 Resources

Resources es un *plugin* de Grails que permite enlazar recursos a la aplicación, permitiendo crear las dependencias que se requieren. Con el uso de esta extensión también se permite la creación de diferentes entornos, especificando los diferentes recursos que necesita cada uno de ellos, consiguiendo con ello una mayor sencillez [30].

2.5 Servidores de aplicaciones

Un servidor de aplicaciones es un programa software que ofrece acceso a los clientes a las aplicaciones desplegadas. El acceso se realiza a través de una conexión a Internet, por medio de un terminal, ya sea un ordenador, un terminal móvil o cualquier otro dispositivo capaz de conectarse. Además, un servidor de aplicaciones es un elemento intermedio que se encarga de proporcionar a los usuarios acceso a los datos, y se encarga de proporcionar mantenimiento y seguridad [31].

2.5.1 Glassfish

Glassfish en un servidor de aplicaciones de código abierto, para aplicaciones J2EE soportado oficialmente por Sun Microsystem [32]. Este es el servidor de aplicaciones del que dispone Salenda para publicar las aplicaciones. Utilizan este servidor, no solo porque está especialmente dedicado para aplicaciones desarrolladas en Java, sino porque también es de código abierto. Además, en este servidor ya cuento con experiencia en su manejo, por lo que no se requiere un tiempo para aprender a utilizarlo.

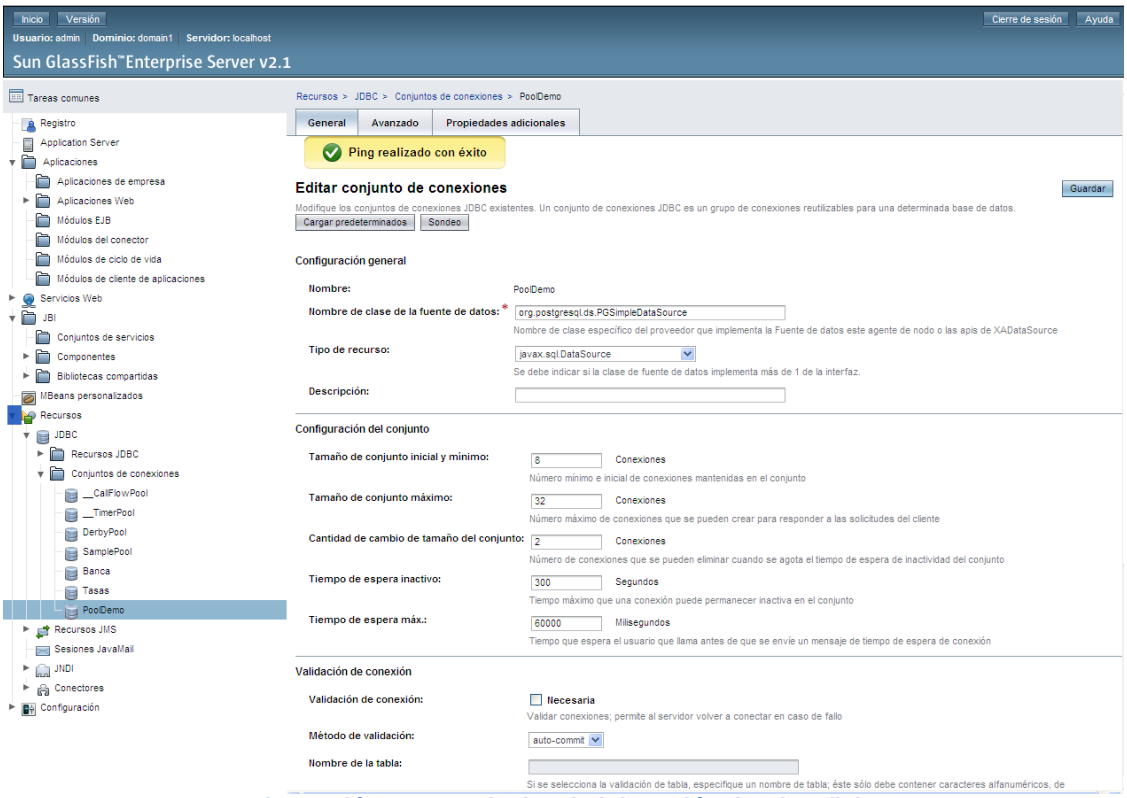

Ilustración 7. Consola de administración de Glassfish

2.6 Entornos de desarrollo integrados (IDE)

Un entorno de desarrollo integrado es un elemento software, que incorpora diferentes herramientas que permiten al programador crear código [33]. Los IDE pueden estar construidos para un solo lenguaje, como por ejemplo JBuilder [34] que se utiliza para desarrollar en el lenguaje Java, o para varios lenguajes como puede ser Eclipse, que da soporte para desarrollar, principalmente para Java, pero también para C, C++, Cobol, PHP, etcétera [35].

2.6.1 SpringSource tool suite

SpringSource Tool Suite (STS) [36] es un IDE basado en Eclipse y extensible mediante *plugins* creado por SpringSource [37]. Este IDE esta disponible para múltiples plataformas, como Windows [38], Linux [39], Mac OS X [40]. STS está orientado al desarrollo de Java, Spring y Groovy y Grails, siento este último el que utilizaremos. STS trae integrado vFabric tc Server Developer Edition [41], un servidor mejor que Apache Tomcat [42], que también es de código abierto. Este servidor ofrece mejoras como la visualización del rendimiento de las aplicaciones, integración con STS, funciones mejoradas de diagnostico de problemas, etcétera.

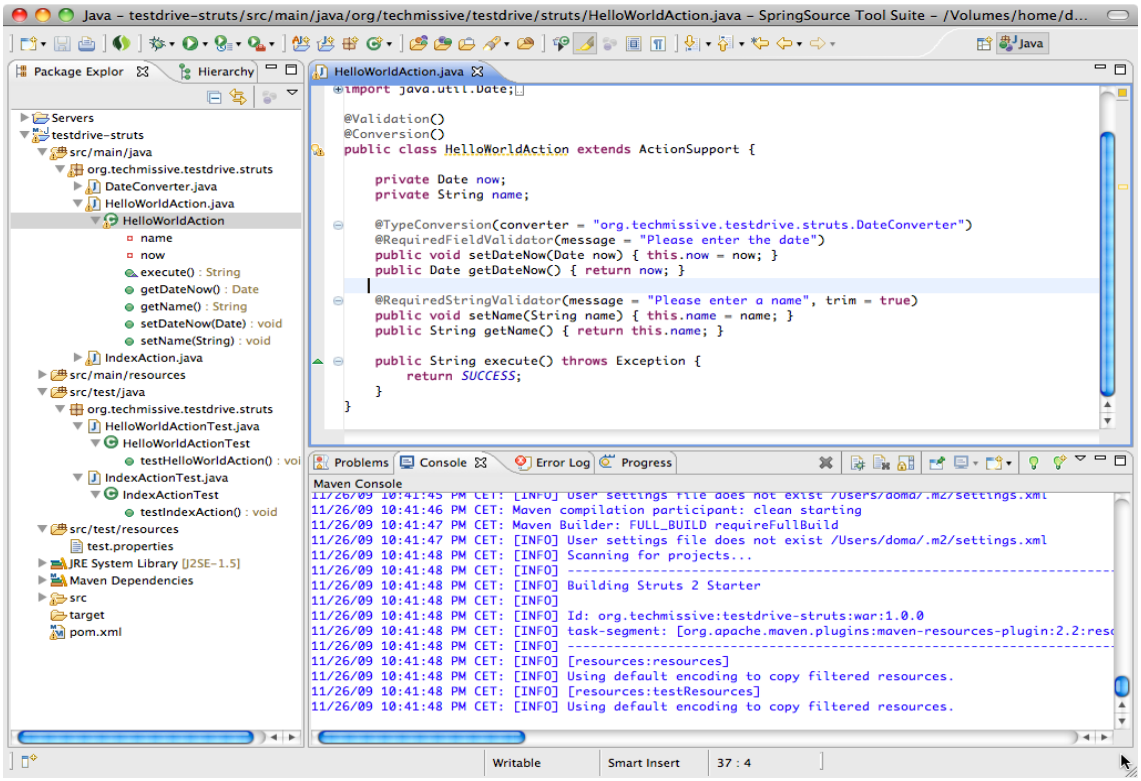

Ilustración 8. SpringSource Tool Suite

También existen otros IDE que ofrecen soporte a Groovy y Grails, como por ejemplo, NetBeans o IntelliJ IDEA [43].

Se ha escogido STS como IDE para el desarrollo de este proyecto, frente a los otros IDE sobre los que se podría realizar el desarrollo, porque tanto Groovy como Grails son productos de SpringSource y STS está creado para potenciar las posibilidades que ofrecen el lenguaje y el *framework* elegidos.

2.7 REST

REST (Representational State Transfer) es un tipo de arquitectura software que permite la gestión de recursos o de información mediante mensajes con formato XML a través del protocolo HTTP. REST permite diferentes operaciones, a través de las opciones que ofrece HTTP [44]:

- **PUT**: esta operación permite insertar contenido.
- **GET**: con esta operación podemos consultar contenido.
- **POST**: permite modificar o actualizar el contenido.
- **DELETE**: esta operación nos permite eliminar el contenido.

Para acceder a los recursos con REST, y puesto que utilizamos HTTP, solo necesitamos especificar la URL del contenido deseado, por ejemplo: http://api.twitter.com/1/followers/1000.xml con la que obtendríamos los *followers* de la cuenta con identificador 1000 en Twitter.

El resultado obtenido, en caso de realizar una consulta, es un documento en formato XML o JSON que contiene la información solicitada:

- XML: son las siglas de eXtensible Markup Lenguage, que en español quiere decir lenguaje extensible de marcado. Es una forma de definir lenguajes, por ejemplo XHTML o SVG. En el siguiente ejemplo podemos ver la estructura de un fichero XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<client>
        <id>80</id>
        <name>Sample name</name>
        ...
        <billing>
                <currency>EUR</currency>
                <tax1>
                        <name>IVA</name>
                        <rate>18.00</rate>
                </tax1>
        </billing>
</client>
```

Ilustración 9. Ejemplo XML

- JSON: son las siglas de JavaScript Object Notation, y es un formato de intercambio de datos utilizado en muchos lenguajes de programación. El ejemplo anteriormente citado, produce un resultado de este estilo en formato JSON, en el que se muestran los identificadores de los *followers* de ese usuario:

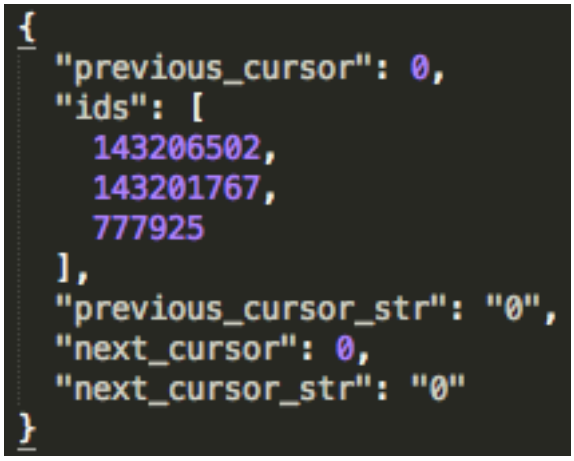

Ilustración 10. Ejemplo resultado REST

El uso de REST está documentado en las interfaces de programación de aplicaciones (API) específicas de cada aplicación. Por ejemplo, podemos encontrar la API de Twitter en [45].

REST puede ser utilizado como un servicio web de operaciones limitadas a las que ofrece el protocolo HTTP, mientras que un servicio web completo, como puede ser SOAP, permite definir las operaciones que el desarrollador crea conveniente. Por eso REST es más rápido y necesita menos configuración que los servicios web.

Existe desde hace años la discusión que enfrenta a REST con los servicios web, comparando la sencillez de uno frente a la versatilidad del otro o la accesibilidad frente a la organización jerarquizada, respectivamente. Ambas tecnologías son utilizadas por grandes empresas en la actualidad, por ejemplo REST es utilizado por Twitter, Yahoo o Ebay, mientras que los servicios web son utilizados en empresas como Amazon. Muchas de ellas ofrecen las dos alternativas a los desarrolladores.

Con el uso de REST se permitirá que desde la aplicación desarrollada se acceda a información ubicada en otras dos aplicaciones:

- **Factura Directa**: de donde se obtendrá información sobre los clientes de Salenda.
- **JIRA**: de donde se obtendrán los datos sobre los diferentes proyectos que se están desarrollando, así como el tiempo empleado en realizar una determinada actividad perteneciente a un proyecto.

2.8 Herramientas de gestión

Para la gestión de las tareas como de la documentación utilizada a lo largo del desarrollo del proyecto, se van a utilizar varias herramientas de Atlassian [46]. El uso de estas herramientas nos permitirá mantener un estricto control del avance el proyecto y mantener organizada toda la información disponible.

2.8.1 JIRA + GreenHopper

JIRA y GreenHopper son dos herramientas de Atlassian que facilitan el desarrollo de proyectos, principalmente informáticos, pero pueden ser de cualquier tipo.

2.8.1.1 JIRA

JIRA es una herramienta web para la gestión de incidencias y tareas, la gestión ágil de proyectos y que cuenta con diagramas de estados personalizados. Además es extensible a través de *plugins*, que permiten incrementar la velocidad del desarrollo de software. Sus principales funcionalidades son:

- **Gestión de incidencias y tareas**: gestión simple y flexible. Permite la integración con el código fuente y con diversos entornos de desarrollo, lo que hace posible la adaptación a la forma en la que trabajas.
- **Desarrollo ágil de software**: con JIRA puedes seguir las diferentes metodologías agiles para el desarrollo del software.
- **Gestión de proyectos**: permite la gestión de varios proyectos en un único lugar.
- **Flujos de trabajo personalizados**: puedes definir los estados por los que pasa cada tarea o problema a solucionar.
- **Informes y análisis**: es posible generar informes al vuelo utilizando JIRA Query Language (JQL). [47]
- ***Plugins* y extensiones**: permiten personalizar y agregar nuevas posibilidades a JIRA. Existen más de 150 *plugins* y extensiones, aunque se pueden desarrollar extensiones personalizadas.

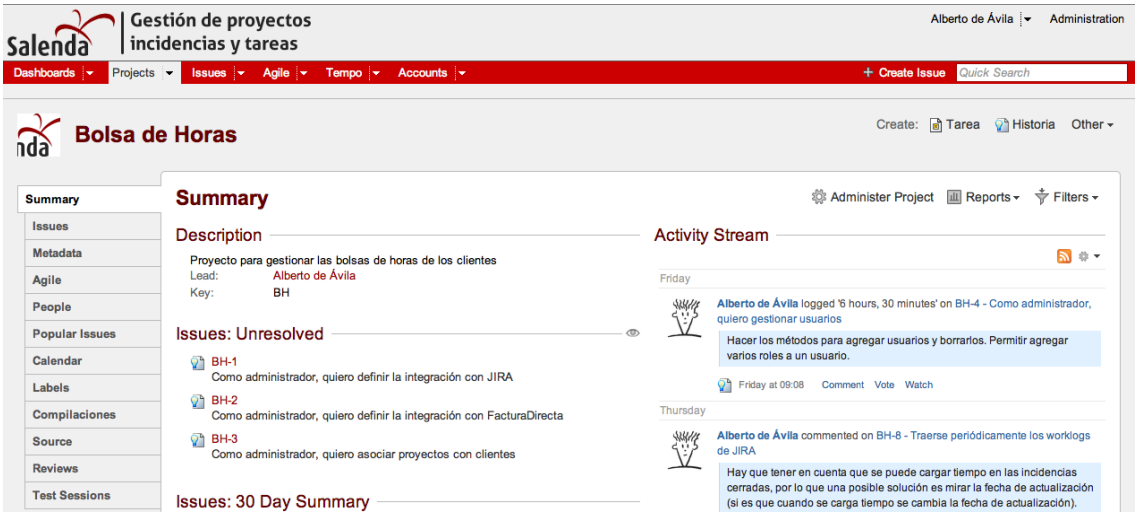

Ilustración 11. Jira

2.8.1.2 GreenHopper

GreenHopper es una herramienta de Atlassian que se integra con JIRA para permitir la gestión de proyectos siguiendo metodologías ágiles. Con esta herramienta es posible crear tarjetas virtuales totalmente configurable, en las que cada tarjeta es una tarea a realizar. Además, la planificación de las tareas en iteraciones se realiza arrastrando las tareas y observa su progreso con pizarras Kanban [48], que permiten, de un vistazo, ver que tareas están en progresa, ver cuales están hechas y cuales aún no han comenzado.

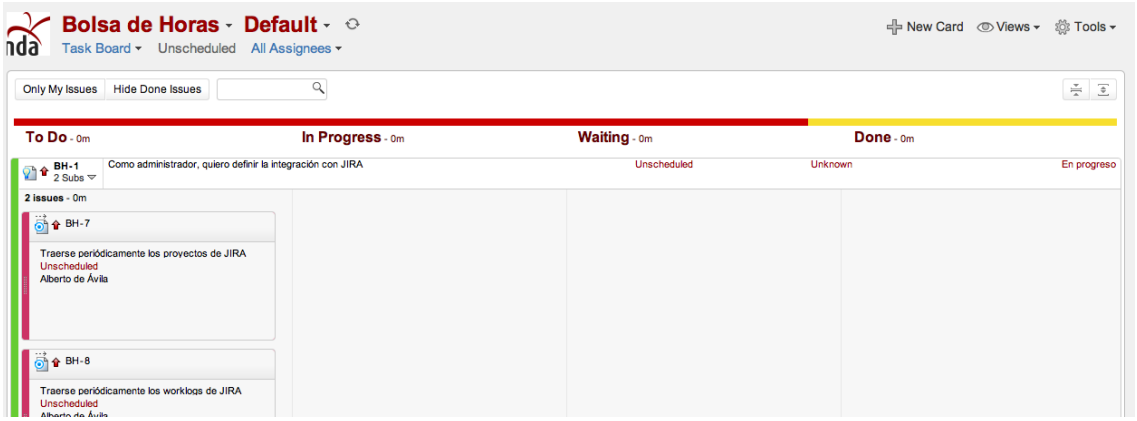

Ilustración 12. GreenHopper

2.8.2 Confluence

Confluence es otra herramienta de Atlassian para la gestión de contenidos y una plataforma de colaboración. Permite crear y compartir contenido con los usuarios a través de una interfaz sencilla de utilizar. Además cuenta con integración con Microsoft Office [49], control de versiones, control de accesos y una extensa librería de *plugins* que permiten agregar nuevas funcionalidades.

Ilustración 13. Confluence

2.9 Otras herramientas

En este apartado se van a especificar las diferentes herramientas empleadas durante el desarrollo del proyecto que no corresponden a ninguna sección de las anteriormente mencionadas.

2.9.1 Firebug

Tango Firebug [50] para Mozilla Firefox [51], como Firebug Lite [52] en su versión para el navegador Google Chome [53], son herramientas que se integran con el navegador y permiten visualizar código HTML [54] de una página web, los estilos de cada elemento HTML, editar estos contenidos dinámicamente y ver el resultado y permite capturar todas las llamadas que se realizan una vez la página está cargada.

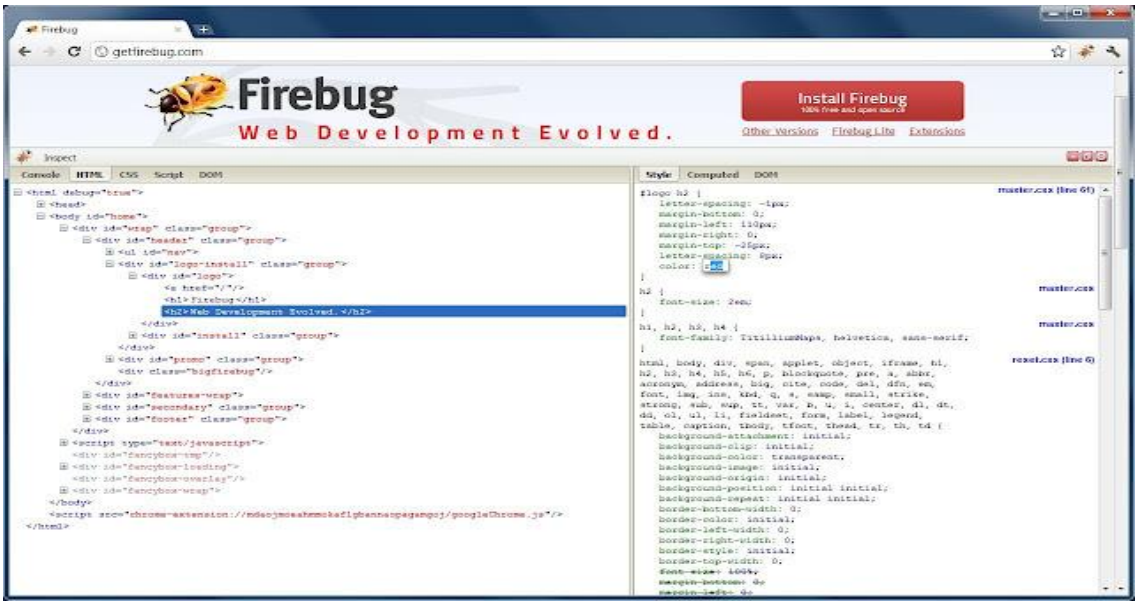

Ilustración 14. Firebug mostrando código HTML y CSS

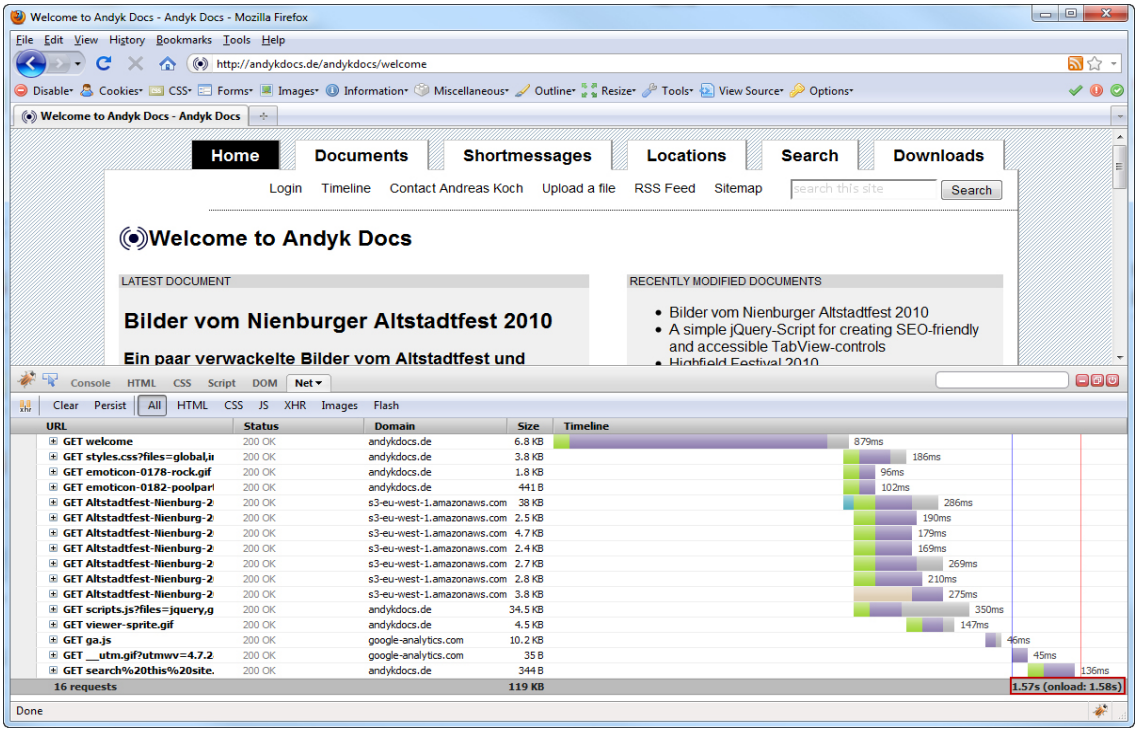

Ilustración 15. Firebug mostrando las llamadas ejecutadas

2.9.2 GIT

GIT es un sistema de control de versiones de código abierto y gratuito, que permite desarrollar un mismo proyecto de forma distribuida en distintos terminales, sin necesidad de realizar copias innecesarias. Basta con crear un repositorio de contenido en el que estarán los proyectos que queramos controlar, subir los proyectos por primera vez y comenzar a utilizar el control de versiones. Es costumbre en uso de sistemas de control de versiones revisar si existen cambios antes de trabajar en algo nuevo. Si es así, debemos actualizar nuestro proyecto para trabajar sobre la versión más reciente. Cada vez que realicemos cambios, debemos subirlos al repositorio de contenido para que quede almacenado.

Con GIT se permite crear diversas ramas, para mantener tanto las versiones estables como el avance en el desarrollo de nuevas funcionalidades aún por acabar [55].

2.10 Google Chart Tools

Google Chart Tools es una herramienta que permite mostrar gráficas en aplicaciones web de forma dinámica. Además es de uso gratuito, aunque limitado a 50.000 peticiones a su API, Google Chart ofrece una gran variedad de tipos de gráfica, configurables de muchas maneras y que ofrecen a los desarrolladores una opción a tener en cuenta a la hora de emplear gráficas en aplicaciones web.

Google Chart Tools funciona mediante Javascript enviando una petición a la API para que devuelva el gráfico y pintarlo en la zona de la página deseada.

En la siguiente imagen podemos ver un ejemplo de como se visualizan las gráficas con esta herramienta:

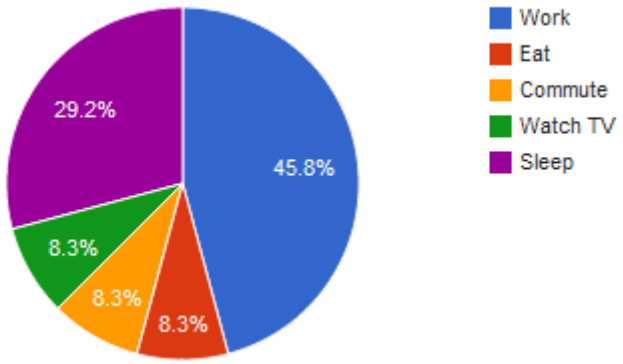

Ilustración 16. Google Chart Tools

Entre los tipos de gráficas que podemos encontrar están:

- Gráfica de barras.
- Gráfica de tarta.
- Gráfica de área.
- Gráfica de columnas.
- Gráfica de mapas.
- Diagrama en forma de árbol.
- Organigrama.
- Etc.

Sección 3

3 Análisis del sistema

A continuación se detallan los diferentes requisitos *software* que presenta la aplicación a desarrollar y sobre la que se centra el presente documento.

3.1 Identificación de escenarios y usuarios

La aplicación que se va a desarrollar consta de una funcionalidad que permita a los clientes poder visualizar las diferentes incidencias que hay en sus proyectos y las horas de soporte restantes. Así mismo se permitirá a los administradores gestionar los diferentes aspectos de la configuración de los que dispone la aplicación.

Se entiende por tanto, que solo existe un escenario posible, con diferentes casos de uso en función del rol dentro de la aplicación.

3.1.1 Usuarios

Dentro de los usuarios vamos a englobar a todos aquellos que utilicen la aplicación, sin distinción de rol, pero existirán dos tipos de usuario, en función de las acciones que puedan realizar:

- **Clientes**: podrán acceder a la información y consultar la bolsa de horas.
- **Administradores**: gestionar los usuarios de la aplicación, la configuración de la misma y gestionar las bolsas de horas.

A continuación se enumeran y detallan las diferentes acciones que pueden realizar cada uno de estos roles.

3.1.1.1 Clientes

Las diferentes acciones que podrán realizar los clientes, una vez se han autenticado en la aplicación son:

- Consultar los diferentes proyectos que está desarrollando o ha desarrollado Salenda.
- Consultar los diferentes movimientos realizados sobre la bolsa de horas de soporte contratada. Estos movimientos pueden ser:
    - Positivos: si se contratan horas o se realiza una devolución de horas.
    - Negativo: si se ha gastado tiempo en solucionar una incidencia.

  Recibirá un correo cuando la bolsa de horas baje de un determinado umbral.

3.1.1.2 Administradores

Los administradores son los que mayor número de acciones podrán realizar. Entre ellas:

- Gestionar los usuarios que acceden a la aplicación y sus datos.

- Gestionar los diferentes parámetros de configuración de la aplicación, para su interacción con otras aplicaciones a través de REST:
    - Factura Directa: nombre de usuario, contraseña y enlace a la aplicación.
    - JIRA: nombre de usuario, contraseña, enlace a la aplicación, consulta de las incidencias que debe obtener de la aplicación y otros valores que permitan ofrecer la información detallada de las incidencias al usuario.
- Gestionar la relación existente entre cliente y proyecto.
- Gestionar la bolsa de horas de los usuarios.

3.2 Establecimiento de requisitos del *software*

En este apartado se van a detallar el conjunto de diferentes requisitos que se van a desarrollar en la aplicación y se detalla la nomenclatura que se va a utilizar. Para los requisitos, se dividen según sean requisitos funcionales y requisitos no funcionales.

Para el desarrollo de aplicaciones informáticas, el análisis típico está descompuesto en requisitos para poder determinar, no solo los costes, sino que también permite una mejor comprensión del alcance de la aplicación y facilita los términos acordados entre las dos partes contratantes, el cliente y la empresa.

Como hemos mencionado anteriormente, los requisitos pueden ser:

- Funcionales: son aquellos requisitos que determinan la funcionalidad de la aplicación y definen todas aquellas acciones que pueden realizar los diferentes roles de la aplicación.
- No funcionales: son los requisitos que determinan las restricciones o condiciones en aspectos de calidad, estándares, costes, estabilidad y portabilidad.

3.2.1 Prototipo de requisito

De cara a especificar el formato, un requisito contará con diferentes campos:

- **Identificador**: incluye el tipo de requisito, y un número identificativo, consecutivo y que comienza en 1. Las numeraciones son dependientes del tipo, y su nomenclatura es:
  - **Funcionales**: F-X, donde X es el número correspondiente.
  - **No funcionales**: NF-Y, donde Y es el número correspondiente.
- **Nombre:** nombre descriptivo del requisito.
- **Prioridad**: define en cual de los siguientes estados se encuentra el requisito, en función de su importancia:
  - **Requerido.**
  - **Deseable.**
  - **Opcional.**
- **Descripción**: se detalla con claridad el objetivo del requisito.
- **Cambios**: registro de cambios en el requisito.

Requisito	**<identificador>**	**Nombre**	**<nombre>**
Prioridad	Requerido ( )	Deseable ( )	Opcional ( )
Descripción	<descripción>		
Cambios	<cambios>		

Tabla 1. Prototipo de requisito

3.2.2 Requisitos funcionales

A continuación se detallan los requisitos funcionales de la bolsa de horas:

Requisito	F-1	Nombre	Alta de usuarios
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )

Descripción	La aplicación permitirá a los administradores crear usuarios indicando:
	• Nombre.
	• Apellidos.
	• Correo electrónico.
	• Contraseña.
	• Rol: pudiendo seleccionar uno o ambos.
	  ○ Administrador.
	  ○ Cliente.		

Cambios	Versión inicial: 22/02/2012		

Tabla 2. Requisito funcional 1 - Alta de usuarios

Requisito	F-2	Nombre	Modificación de usuarios
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá a los administradores modificar los datos de los usuarios.		
Cambios	Versión inicial: 22/02/2012		

Tabla 3. Requisito funcional 2 - Modificación de usuarios

Requisito	F-3	Nombre	Borrado de usuarios
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá a un administrador borrar usuarios.		
Cambios	Versión inicial: 22/02/2012		

Tabla 4. Requisito funcional 3 - Borrado de usuarios

Requisito	F-4	Nombre	Mostrar usuarios
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación mostrará a los administradores de la aplicación, el listado completo de usuarios de la aplicación.		
Cambios	Versión inicial: 22/02/2012		

Tabla 5. Requisito funcional 4 - Mostrar usuarios

Requisito	F-5	Nombre	Agregar datos de integración con Factura Directa
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá a los administradores definir los datos para la interactuación con Factura Directa. Los datos a definir son: • URL de Factura Directa. • Clave de autenticación proporcionada por Factura Directa. • Contraseña de la cuenta de Factura Directa.		
Cambios	Versión inicial: 22/02/2012		

Tabla 6. Requisito funcional 5 - Agregar integración Factura Directa

Requisito	F-6	Nombre	Modificar datos integración con Factura Directa
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá a los administradores modificar los datos de integración con Factura Directa.		
Cambios	Versión inicial: 22/02/2012		

Tabla 7. Requisito funcional 6 - Modificar integración Factura Directa

Requisito	F-7	Nombre	Borrar los datos de integración con Factura Directa
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá a los administradores borrar los datos de integración con Factura Directa.		
Cambios	Versión inicial: 22/02/2012		

Tabla 8. Requisito funcional 7 - Borrar integración Factura Directa

Requisito	F-8	Nombre	Listar datos integración con Factura Directa
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá a los administradores visualizar los datos de integración con Factura Directa.		
Cambios	Versión inicial: 25/02/2012		

Tabla 9. Requisito funcional 8 - Listar datos integración Factura Directa

Requisito	F-9	Nombre	Obtener los datos de los clientes de Factura Directa
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá obtener periódicamente los datos de los clientes presentes en Factura Directa. Los datos serán obtenidos mediante el uso de REST.		
Cambios	Versión inicial: 22/02/2012		

Tabla 10. Requisito funcional 9 - Obtener datos clientes Factura Directa

Requisito	F-10	Nombre	Relacionar datos de clientes con clientes
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá a los administradores relacionar clientes de la aplicación con los datos provenientes de Factura Directa.		
Cambios	Versión inicial: 22/02/2012		

Tabla 11. Requisito funcional 10 - Relacionar datos de clientes con clientes

Requisito	F-11	Nombre	Agregar datos integración con JIRA
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá a los administradores definir los datos para la interactuación con JIRA. Los datos a definir son: • URL de JIRA. • Nombre de usuario de JIRA. • Contraseña de JIRA. • Consulta para traer los datos de JIRA.		
Cambios	Versión inicial: 22/02/2012		

Tabla 12. Requisito funcional 11 - Agregar datos de integración JIRA

Requisito	F-12	Nombre	Modificar datos integración con JIRA
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá a los administradores modificar los datos de integración con JIRA.		
Cambios	Versión inicial: 22/02/2012		

Tabla 13. Requisito funcional 12 - Modificar datos de integración JIRA

Requisito	F-13	Nombre	Borrar los datos de integración con JIRA
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá a los administradores borrar los datos de integración con JIRA.		
Cambios	Versión inicial: 22/02/2012		

Tabla 14. Requisito funcional 13 - Borrar datos de integración JIRA

Requisito	F-14	Nombre	Listar los datos de integración con JIRA
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá a los administradores visualizar los datos de integración con JIRA.		
Cambios	Versión inicial: 25/02/2012		

Tabla 15. Requisito funcional 14 - Listar datos de integración JIRA

Requisito	F-15	Nombre	Obtener los proyectos de JIRA
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá obtener, periódicamente, los proyectos de JIRA. Los datos serán obtenidos mediante el uso de REST.		
Cambios	Versión inicial: 22/02/2012		

Tabla 16. Requisito funcional 15 - Obtener proyectos de JIRA

Requisito	F-16	Nombre	Crear una bolsa a un cliente
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá a los administradores crear bolsas a los clientes.		
Cambios	Versión inicial: 22/02/2012		

Tabla 17. Requisito funcional 16 - Crear una bolsa a un cliente

Requisito	F-17	Nombre	Borrar bolsas de un cliente
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá a los administradores borrar bolsas de los clientes.		
Cambios	Versión inicial: 22/02/2012		

Tabla 18. Requisito funcional 17 - Borrar bolsas de un cliente

Requisito	F-18	Nombre	Relacionar proyectos con clientes
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá a los administradores relacionar a un proyecto con una bolsa de un cliente.		
Cambios	Versión inicial: 22/02/2012		

Tabla 19. Requisito funcional 18 - Relacionar proyectos con clientes

Requisito	F-19	Nombre	Obtener el tiempo empleado por tarea de JIRA
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá obtener, periódicamente, los tiempos empleados en resolver las tareas de un proyecto presente en JIRA. Los datos serán obtenidos mediante el uso de REST.		
Cambios	Versión inicial: 22/02/2012		

Tabla 20. Requisito funcional 19 - Obtener tiempos por tarea de JIRA

Requisito	F-20	Nombre	Agregar movimiento a la bolsa de horas
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá a los administradores agregar un movimiento a una bolsa de horas asociada a un cliente. Cada movimiento tendrá: • Descripción: incidencia a la que se asocia un movimiento o descripción de la situación. • Tiempo empleado. Puede ser positivo si el		
Cambios	Versión inicial: 22/02/2012		

Tabla 21. Requisito funcional 20 - Agregar movimientos a la bolsa de horas

Requisito	F-21	Nombre	Reembolsar movimiento de la bolsa de horas
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá a los administradores reembolsar un movimiento de una bolsa de horas asociada a un cliente.		
Cambios	Versión inicial: 22/02/2012		

Tabla 22. Requisito funcional 21 - Borrar movimientos de la bolsa de horas

Requisito	F-22	Nombre	Listar movimiento de la bolsa de horas
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá a los administradores visualizar los movimientos de una bolsa de horas asociada a un cliente.		
Cambios	Versión inicial: 22/02/2012		

Tabla 23. Requisito funcional 22 - Listar movimientos de bolsa de horas

Requisito	F-23	Nombre	Ver todas las bolsas de horas
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá a los administradores ver cualquier bolsa de horas.		
Cambios	Versión inicial: 22/02/2012		

Tabla 24. Requisito funcional 23 - Ver todas las bolsas de horas

Requisito	F-24	Nombre	Ver movimientos de horas asociada al cliente
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá a los clientes ver los movimientos de la bolsa de horas que tienen asociada.		
Cambios	Versión inicial: 22/02/2012		

Tabla 25. Requisito funcional 24 - Ver bolsa de horas asociada a un cliente

Requisito	F-25	Nombre	Notificaciones de la bolsa de horas
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación enviará una notificación a los clientes cuando su bolsa de horas este por debajo de un limite.		
Cambios	Versión inicial: 22/02/2012		

Tabla 26. Requisito funcional 25 - Notificaciones de la bolsa de horas

Requisito	F-26	Nombre	Acceder a la aplicación
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación permitirá el acceso a aquellos usuarios que se autentiquen introduciendo: • Correo electrónico. • Contraseña.		
Cambios	Versión inicial: 22/02/2012		

Tabla 27. Requisito funcional 26 - Acceder a la aplicación

Requisito	F-27	Nombre	Ver estadísticas por tipo de tarea
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	Como usuario quiero ver las estadísticas de mis movimientos por tipo de tarea.		
Cambios	Versión inicial: 13/05/2012		

Tabla 28. Requisito funcional 27 - Estadísticas por tipo de tarea

Requisito	F-28	Nombre	Ver estadísticas por tiempo gastado
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	Como usuario quiero ver las estadísticas de mis movimientos por tiempo gastado.		
Cambios	Versión inicial: 13/05/2012		

Tabla 29. Requisito funcional 28 - Estadísticas por tiempo gastado

Requisito	F-29	Nombre	Ver precios de las bolsas de horas
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	Como usuario quiero ver los precios de las bolsas.		
Cambios	Versión inicial: 13/05/2012		

Tabla 30. Requisito funcional 29 - Precios de las bolsas de horas

Requisito	F-30	Nombre	Enviar correo al crear usuario
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación enviará un correo cuando se cree un nuevo usuario, especificando: • Nombre de usuario. • Contraseña.		
Cambios	Versión inicial: 13/05/2012		

Tabla 31. Requisito funcional 30 - Enviar correo al crear usuario

3.2.3 Requisitos no funcionales

A continuación se detallan aquellos requisitos de tipo no funcional:

Requisito	NF-1	Nombre	Funcionamiento con internet
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	Para acceder a la aplicación será necesaria una conexión a internet.		
Cambios	Versión inicial: 22/02/2012		

Tabla 32. Requisito no funcional 1 - Funcionamiento con internet

Requisito	NF-2	Nombre	Funcionamiento con un navegador web
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	Para acceder a la aplicación será necesario un navegador web.		
Cambios	Versión inicial: 22/02/2012		

Tabla 33. Requisito no funcional 2 - Funcionamiento con un navegador web

Requisito	NF-3	Nombre	Compatibilidad con Mozilla Firefox
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación será compatible con el navegador Mozilla Firefox versión 3 o superior.		
Cambios	Versión inicial: 22/02/2012		

Tabla 34. Requisito no funcional 3 - Compatibilidad con Firefox

Requisito	NF-4	Nombre	Compatibilidad con Google Chrome
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación será compatible con el navegador Google Chrome versión 2 o superior.		
Cambios	Versión inicial: 22/02/2012		

Tabla 35. Requisito no funcional 4 - Compatibilidad con Chrome

Requisito	NF-5	Nombre	Compatibilidad con Safari
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación será compatible con el navegador Safari versión 3 o superior.		
Cambios	Versión inicial: 22/02/2012		

Tabla 36. Requisito no funcional 5 - Compatibilidad con Safari

Requisito	NF-6	Nombre	Compatibilidad con Internet Explorer
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación será compatible con el navegador Internet Explorer versión 8 o superior.		
Cambios	Versión inicial: 22/02/2012		

Tabla 37. Requisito no funcional 6 - Compatibilidad con IE

Requisito	NF-7	Nombre	*Framework*
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación será desarrollada con el *famework* Grails.		
Cambios	Versión inicial: 22/02/2012		

Tabla 38. Requisito no funcional 7 - *Framework*

Requisito	NF-8	Nombre	Lenguaje de programación
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación será desarrollada con el Groovy como lenguaje de programación.		
Cambios	Versión inicial: 22/02/2012		

Tabla 39. Requisito no funcional 8 - Lenguaje de programación

Requisito	NF-9	Nombre	IDE
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación será desarrollada SpringSource Tool Suite como IDE.		
Cambios	Versión inicial: 22/02/2012		

Tabla 40. Requisito no funcional 9 - IDE

Requisito	NF-10	Nombre	Base de datos
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación utilizará como una base de datos H2.		
Cambios	Versión inicial: 22/02/2012		

Tabla 41. Requisito no funcional 10 - Base de datos

Requisito	NF-11	Nombre	Servidor
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	La aplicación usará Glassfish como servidor.		
Cambios	Versión inicial: 22/02/2012		

Tabla 42. Requisito no funcional 11 - Servidor

Requisito	NF-12	Nombre	Control de versiones
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	Para el desarrollo de la aplicación se utilizará GIT para el control de versiones.		
Cambios	Versión inicial: 22/02/2012		

Tabla 43. Requisito no funcional 12 - Control de versiones

Requisito	NF-13	Nombre	Manual de usuario
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	Se proporcionará un manual de uso de la aplicación para usuarios y para administradores.		
Cambios	Versión inicial: 22/02/2012		

Tabla 44. Requisito no funcional 13 - Manual de usuario

Requisito	NF-14	Nombre	Gestión del desarrollo de la aplicación
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	Para el desarrollo de la aplicación se utilizará JIRA como herramienta para gestionar el avance los requisitos.		
Cambios	Versión inicial: 22/02/2012		

Tabla 45. Requisito no funcional 14 - Gestión del desarrollo de la aplicación

Requisito	NF-15	Nombre	Asignación de proyectos mediante *drag and drop*
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	Para la asignación de proyectos se utilizará la técnica de *drag and drop*, que permitirá al administrador decidir en qué bolsa sitúa un proyecto.		
Cambios	Versión inicial: 13/05/2012		

Tabla 46. Requisito no funcional 15 - Proyectos con *drag and drop*

Requisito	NF-16	Nombre	Edición de la información
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	Para editar la información se utilizará un *plugin* de jQuery (jEditable) que permite al administrador editar cualquier campo marcando el campo a editar, cambiando la información y pulsando la tecla *enter*.		
Cambios	Versión inicial: 13/05/2012		

Tabla 47. Requisito no funcional 16 - Edición de la información

Requisito	NF-17	Nombre	Herramienta para mostrar estadísticas
Prioridad	Requerido (*)	Deseable ( )	Opcional ( )
Descripción	Para mostar la información sobre las estadísticas se utilizará Google Chart Tool.		
Cambios	Versión inicial: 13/05/2012		

Tabla 48. Requisito no funcional 17 - Herramienta para mostrar estadísticas

3.3 Especificación de casos de uso

Un caso de uso es un posible uso de la aplicación en una determinada situación, llevada a cabo por un tipo de usuario [56]. El uso de estos diagramas nos facilita el desarrollo de la aplicación porque muestra, de forma clara y concisa, las diferentes acciones que va a permitir la aplicación y como se relacionan entre ellas y qué actores las pueden realizar. Por lo tanto, un diagrama de casos de uso es la representación del caso de uso y consta de:

- **Actor** (o tipo de usuario): representa el tipo de usuario de la aplicación que está realizando las acciones.
- **Casos de uso**: es la tarea realizada por el actor o por otro caso de uso dentro del sistema.
- **Relaciones**: es el nexo entre los actores y los casos de uso con otros casos de uso. Hay varios tipos:
    - **Asociación**: representa la posibilidad de que un usuario realice un caso de uso.
    - **Dependencia**: representa la relación entre dos casos de uso.
    - **Generalización**: representa el uso o la herencia entre casos de uso.
- **Escenario**: nombre del contexto de los casos de uso.

Para la representación de los diagramas se va a seguir el siguiente esquema:

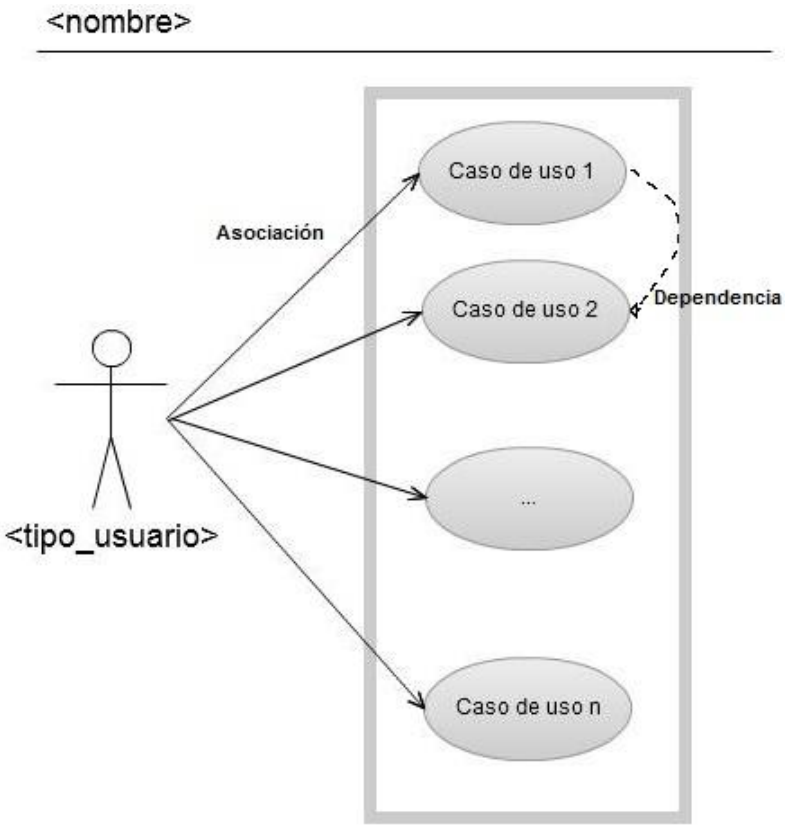

Ilustración 17. Nomenclatura diagrama de casos de uso

Para detallar los diagramas de casos de uso se van a utilizar tablas con la siguiente información:

- Identificador del caso de uso: estará compuesto por la primera letra del actor correspondiente, y un número único para cada actor, por ejemplo S1.
- Nombre del caso de uso.
- Actor que realiza la acción.
- Objetivo de la acción.
- Precondiciones: condiciones previas para poder realizar la acción.
- Postcondiciones: condiciones posteriores para realizar la acción correctamente.
- Escenario básico: detalle de los pasos a seguir para realizar la acción.

Identificador	<identificador>
Nombre	<nombre>
Actor	<actor>
Objetivo	<objetivo>
Precondiciones	<precondiciones>
Postcondiciones	<postcondiciones
Escenario básico	<escenario_basico>

Tabla 49. Nomenclatura descripción casos de uso

A continuación se detallan los diagramas de casos de uso de los dos actores de la aplicación, el cliente y el administrador, y un actor adicional que representa a la aplicación, puesto que esta realiza tareas de forma autónoma.

3.3.1 Sistema

El diagrama de casos de uso del administrador es el siguiente:

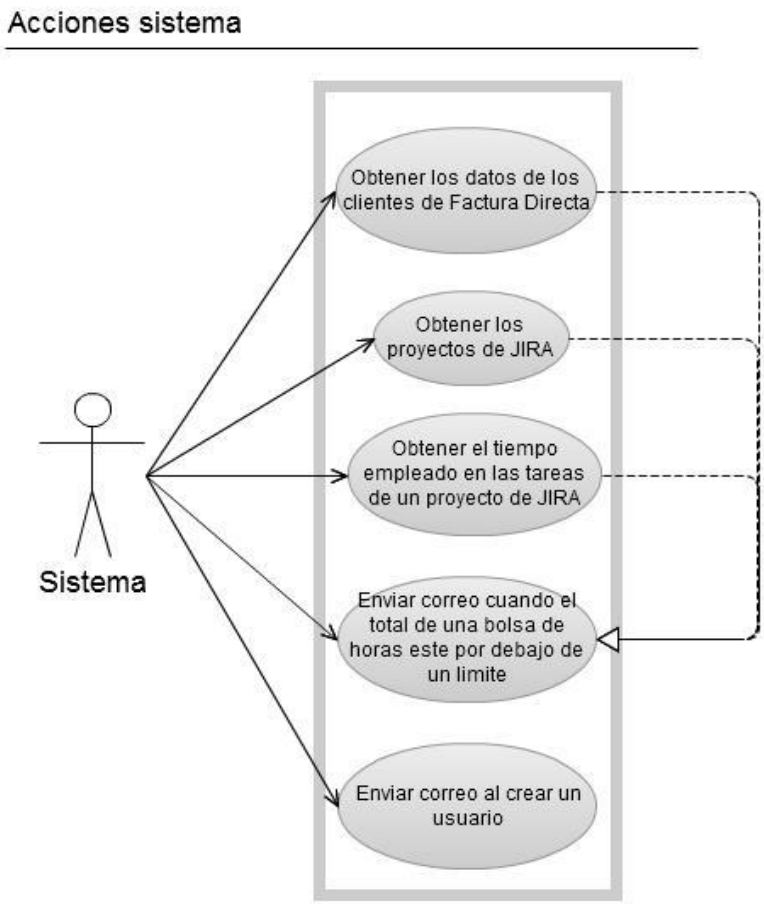

Ilustración 18. Diagrama de casos de uso del sistema

La descripción de los casos de uso del sistema es:

- Obtener periódicamente los datos de los clientes de Factura Directa:

Identificador	S1
Nombre	Obtener datos de Factura Directa.
Actor	Sistema
Objetivo	Conseguir los datos de los clientes presentes en la aplicación Factura Directa para relacionarlos con los usuarios de la aplicación y con los proyectos de JIRA.
Precondiciones	Contar con los datos de acceso a Factura Directa o introducirlos.
Postcondiciones	N / A
Escenario básico	1. Un administrador accede a la aplicación autenticándose con su correo y su contraseña. 2. Introduce los datos para la integración con Factura Directa si no están. 3. La aplicación obtendrá, cuando le corresponda a la tarea periódica, los datos.

Tabla 50. Descripción del caso de uso - Obtener datos de Factura Directa

- Obtener periódicamente los proyectos de JIRA:

Identificador	S2
Nombre	Obtener proyectos de JIRA.
Actor	Sistema
Objetivo	Conseguir los proyectos de los clientes presentes en JIRA para relacionarlos con los usuarios de la aplicación y con los tiempos empleados en resolver tareas.
Precondiciones	Contar con los datos de acceso a JIRA o introducirlos.
Postcondiciones	N / A
Escenario básico	1. Un administrador accede a la aplicación autenticándose con su correo y su contraseña. 2. Introduce los datos para la integración JIRA si no están. 3. La aplicación obtendrá, cuando le corresponda a la tarea periódica, los proyectos.

Tabla 51. Descripción del caso de uso - Obtener proyectos de JIRA

- Obtener periódicamente el tiempo empleado en las tareas de un proyecto de JIRA:

Identificador	S3
Nombre	Obtener tiempos por tarea de JIRA.
Actor	Sistema
Objetivo	Conseguir los tiempos empleados en las tareas de un proyecto de JIRA para relacionarlos con los usuarios de la aplicación y con proyectos.
Precondiciones	Contar con los datos de acceso a JIRA o introducirlos.
Postcondiciones	N / A
Escenario básico	1. Un administrador accede a la aplicación autenticándose con su correo y su contraseña. 2. Introduce los datos para la integración JIRA si no están. 3. La aplicación obtendrá, cuando le corresponda a la tarea periódica, los tiempos por tarea.

Tabla 52. Descripción del caso de uso - Obtener tiempos por tarea

- Enviar correo cuando el total de la bolsa de horas este por debajo de un límite:

Identificador	S4
Nombre	Enviar correo cuando queden pocas horas en la bolsa.
Actor	Sistema
Objetivo	Avisar al cliente para que sepa que quedan pocas horas de su bolsa de soporte.
Precondiciones	Contar con el correo electrónico valido del usuario.
Postcondiciones	N / A
Escenario básico	Cuando una bolsa de horas este por debajo del limite, la aplicación enviara el correo automático.

Tabla 53. Descripción del caso de uso - Enviar correo de aviso

- Enviar correo al crear un usuario:

Identificador	S5
Nombre	Enviar correo cuando se cree un nuevo usuario.
Actor	Sistema
Objetivo	Avisar al cliente para que sepa conozca que se ha creado su usuario en la aplicación y que conozca el nombre de usuario y la contraseña.
Precondiciones	Contar con el correo electrónico valido del usuario.
Postcondiciones	N / A
Escenario básico	1. Ir al apartado de gestión de usuarios. 2. Pulsar sobre el botón de crear un nuevo usuario. 3. Rellenar los datos. 4. Pulsar sobre el botón de crear.

Tabla 54. Descripción del caso de uso - Enviar correo de creación de usuario

3.3.2 Administrador

El diagrama de casos de uso del administrador es el siguiente:

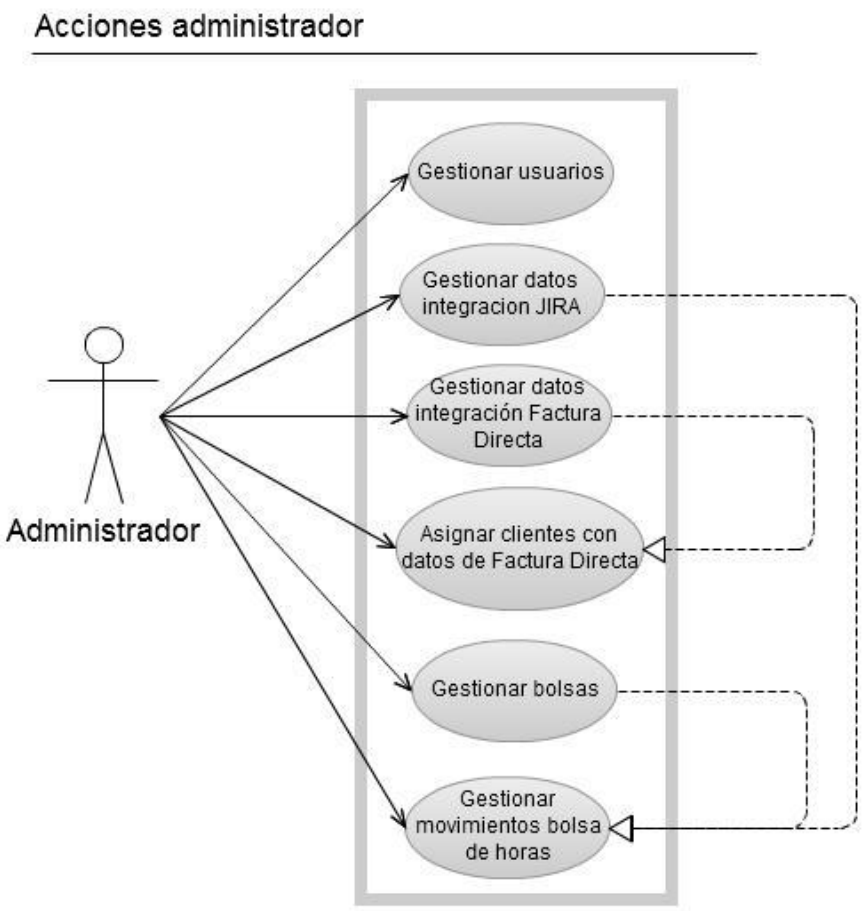

Ilustración 19. Diagrama de casos de uso del administrador

La descripción de los casos de uso del administrador es:

- Gestionar usuarios:

Identificador	A1
Nombre	Gestionar usuarios.
Actor	Administrador
Objetivo	Permitir al administrador crear, borrar, modificar y listar usuarios de la aplicación.
Precondiciones	Acceder a la aplicación y autenticarse con una cuenta valida de administrador
Postcondiciones	N / A
Escenario básico	1. Un administrador accede a la aplicación autenticándose con su correo y su contraseña. 2. Accede a la sección de gestión de usuarios. 3. Realiza la operación pertinente.

Tabla 55. Descripción del caso de uso - Gestionar usuarios

- Gestionar datos de integración con JIRA:

Identificador	A2
Nombre	Gestionar datos de integración con JIRA.
Actor	Administrador
Objetivo	Permitir al administrador crear, borrar, modificar y listar los datos de integración con JIRA.
Precondiciones	Acceder a la aplicación y autenticarse con una cuenta valida de administrador
Postcondiciones	N / A
Escenario básico	1. Un administrador accede a la aplicación autenticándose con su correo y su contraseña. 2. Accede a la sección de configuración. 3. Realiza la operación pertinente.

Tabla 56. Descripción del caso de uso - Gestionar integración con JIRA

- Gestionar datos de integración con Factura Directa:

Identificador	A3
Nombre	Gestionar datos de integración con Factura Directa.
Actor	Administrador
Objetivo	Permitir al administrador crear, borrar, modificar y listar los datos de integración con Factura Directa.
Precondiciones	Acceder a la aplicación y autenticarse con una cuenta valida de administrador
Postcondiciones	N / A
Escenario básico	1. Un administrador accede a la aplicación autenticándose con su correo y su contraseña. 2. Accede a la sección de configuración. 3. Realiza la operación que corresponda.

Tabla 57. Descripción del caso de uso - Gestionar integración con FD

- Asignar clientes con datos que provienen de Factura Directa:

Identificador	A4
Nombre	Asignar los datos que provienen de Factura Directa a los clientes de la aplicación.
Actor	Administrador
Objetivo	Permitir al administrador asignar los datos de Factura directa con los clientes de la aplicación.
Precondiciones	Acceder a la aplicación y autenticarse con una cuenta valida de administrador

Postcondiciones	N / A
Escenario básico	1. Un administrador accede a la aplicación autenticándose con su correo y su contraseña. 2. Accede a la sección de asignación. 3. Realiza la operación pertinente.

Tabla 58. Descripción del caso de uso - Asignar clientes con datos de FD

- Gestionar bolsas de clientes:

Identificador	A5
Nombre	Gestionar bolsas de clientes
Actor	Administrador
Objetivo	Permitir al administrador crear, y borrar bolsas que tiene un cliente.
Precondiciones	Acceder a la aplicación y autenticarse con una cuenta valida de administrador
Postcondiciones	N / A
Escenario básico	1. Un administrador accede a la aplicación autenticándose con su correo y su contraseña. 2. Accede a la sección de bolsas. 3. Crear o borrar la bolsa de un cliente determinado.

Tabla 59. Descripción del caso de uso - Gestionar bolsas de clientes

- Gestionar movimientos de las bolsas de horas:

Identificador	A6
Nombre	Gestionar movimientos de la bolsa de horas
Actor	Administrador
Objetivo	Permitir al administrador crear, borrar, modificar y listar los movimientos de una bolsa de horas.
Precondiciones	Acceder a la aplicación y autenticarse con una cuenta valida de administrador
Postcondiciones	N / A
Escenario básico	4. Un administrador accede a la aplicación autenticándose con su correo y su contraseña. 5. Accede a la sección de bolsas. 6. Acceder a la bolsa sobre la que se quiere hacer la modificación. 7. Realizar la operación correspondiente.

Tabla 60. Descripción del caso de uso - Gestionar movimientos de la bolsa

3.3.3 Cliente

La descripción de los casos de uso que tiene el cliente son:

- Consultar movimientos de las bolsas de horas:

Identificador	C1
Nombre	Consultar movimientos de las bolsa de horas
Actor	Cliente
Objetivo	Que el cliente pueda ver cuantas horas le restan de sus bolsas de horas, en que se han ido gastando las horas y cuantas por cada tarea.
Precondiciones	Que el usuario este dado de alta en la aplicación, tenga al menos una bolsa con un proyecto con al menos una tarea y que se haya dedicado tiempo a esa tarea. Como alternativa también basta con tener un proyecto asignado en una bolsa y haber contratado una bolsa de horas (por lo que se genera un movimiento de la bolsa de horas y se tiene saldo positivo).
Postcondiciones	N / A
Escenario básico	1. Un cliente accede a la aplicación autenticándose con su correo y su contraseña. 2. Se dirige a la sección correspondiente a su bolsa de horas 3. Seleccionar la bolsa de horas que desea ver. 4. Ver los movimientos.

Tabla 61. Descripción del caso de uso - Consultar movimientos

- Ver estadísticas de los movimientos:

Identificador	C2
Nombre	Ver estadísticas de los movimientos
Actor	Cliente
Objetivo	Que el cliente pueda ver estadísticas de los tipos de tarea de sus bolsas y también estadísticas de cuanto tiempo se ha tardado en solucionar una tarea.
Precondiciones	Que el usuario este dado de alta en la aplicación, tenga al menos una bolsa con un proyecto con al menos una tarea y que se haya dedicado tiempo a esa tarea.
Postcondiciones	N / A
Escenario básico	1. Un cliente accede a la aplicación autenticándose con su correo y su contraseña. 2. Se dirige a la sección de estadísticas. 3. Observa las estadísticas.

Tabla 62. Descripción del caso de uso - Ver estadísticas de los movimientos

- Ver listado de precios de las bolsas de horas:

Identificador	C2
Nombre	Ver listado de precios de las bolsas de horas
Actor	Cliente
Objetivo	Que el cliente pueda ver los precios de las bolsas de horas que puede contratar.
Precondiciones	Que el usuario este dado de alta en la aplicación.
Postcondiciones	N / A
Escenario básico	1. Un cliente accede a la aplicación autenticándose con su correo y su contraseña. 2. Se dirige a la sección de precios. 3. Observa el listado de precios.

Tabla 63. Descripción del caso de uso - Ver listado de precios de bolsas

El diagrama de casos de uso del cliente es el siguiente:

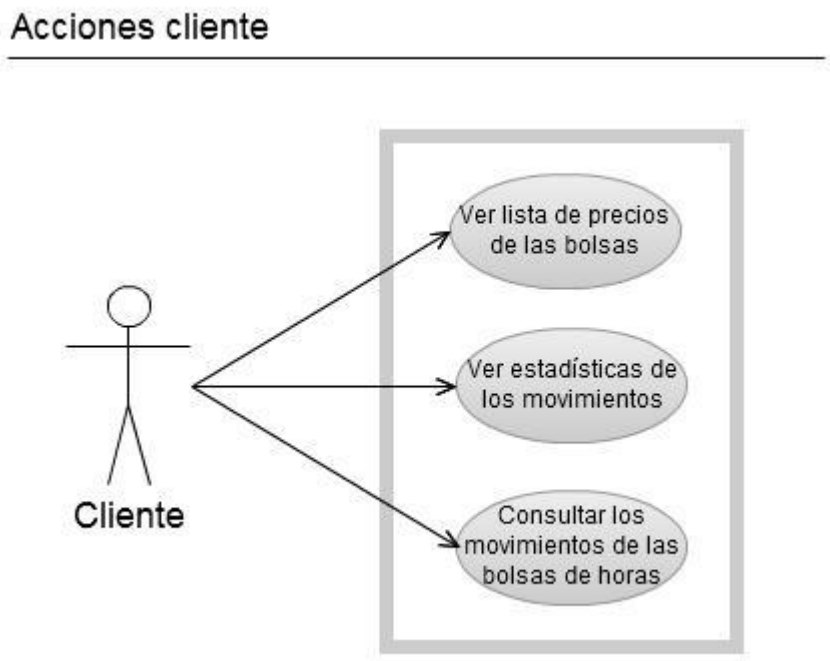

Ilustración 20. Diagrama de casos de uso del cliente

Sección 4

4 Gestión del proyecto

En esta sección se va a detallar los diferentes aspectos de la gestión del proyecto, es decir, los pasos que hay que seguir para el desarrollo de las diferentes tareas. Además se incluye la planificación inicial y la planificación final, así como el presupuesto calculado para llevar a cabo esta aplicación.

4.1 Metodología empleada

Como se ha mencionado en anteriores apartados, se va a utilizar Scrum como metodología para el desarrollo de la aplicación. Scrum es una metodología orientada a proyectos que:

- Tienen una alta complejidad.
- Los requisitos son cambiantes y el proyecto debe ser flexible a los cambios.
- El personal asignado al proyecto es cambiante.
- Cuando los plazos de entrega se dilatan, la calidad de la aplicación no es aceptable o los costes se disparan.

Los beneficios que Scrum ofrece son:

- Otorga a los integrantes del equipo de desarrollo autonomía para realizar las tareas que quieran, dentro de aquellas que se deben realizar en ese *sprint*.
- Permite al cliente observar los avances del proyecto, por lo que los cambios que pueda pedir son menos drásticos.
- En determinados casos, las entregas permiten que la aplicación pueda estar en producción, mientras se continúa el desarrollo. Cuando una entrega se completa y es aprobada por el cliente, puede agregarse a la versión en producción y no esperar a tener el desarrollo completo. Esto es posible en aplicaciones o desarrollos que están muy modularizados.

Scrum utiliza un ciclo de vida iterativo e incremental, lo que significa que cada *sprint* es una iteración en la que se realizan las mismas etapas de desarrollo, implementación y pruebas, de los requisitos obtenidos del análisis. El conjunto de requisitos, que se denomina *product backlog*, se ordena por orden de prioridad para el cliente y para cada *sprint* se decide cuantos de estos requisitos se van a realizar, lo que conforma el *sprint backlog*. De esta forma, cada *sprint* contará con sus pruebas, lo que garantizará el correcto funcionamiento de los requisitos implementados, y cómo funcionan con los anteriormente desarrollados.

Para llevar a cabo la metodología se van a utilizar dos herramientas, JIRA y GreenHopper, que nos facilitarán observar el estado del proyecto, del *sprint* actual, cuantas horas estimadas tiene una tarea, cuantas horas lleva consumidas esa tarea, qué tareas están ahora mismo en progreso y cuales están hechas.

Los pasos durante el desarrollo del proyecto serán:

1. Registrar en JIRA las tareas a realizar.
2. Asignar a cada tarea un tiempo estimado de duración.
3. Priorizar las tareas según el valor que tienen para el cliente y repartir las tareas en *sprints*.

4. Comenzar el *sprint* correspondiente y de las tareas de ese sprint, escoger una y trabajar en ella.
5. Se trabaja diariamente en ella y todos los días tiene lugar una reunión entre el equipo de desarrollo y el Scrum Master. En estas reuniones (*daily meeting*), los miembros del equipo de desarrollo comentan en que han trabajado, en que van a trabajar y los problemas encontrados. El Scrum Master intentará ayudar a solventar los problemas para conseguir que cada desarrollador pueda avanzar de forma ágil.
6. Una vez solucionada la tarea, se realizan las pruebas para verificar que se ha terminado de acuerdo a los requisitos establecidos en el análisis y que todo lo desarrollado hasta el momento funciona correctamente.
7. Se vuelve al paso 4 hasta que se termina el *sprint* y una vez acabado, se comienza el siguiente hasta finalizar el desarrollo. Cada vez que se termina el *sprint*, se muestra el avance al cliente.

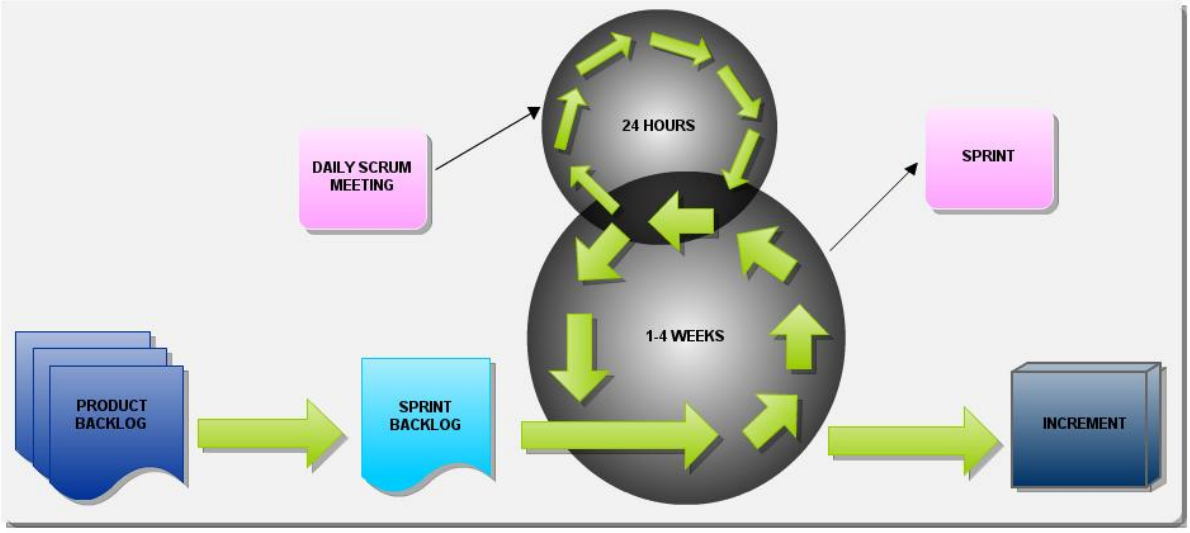

Ilustración 21. Proceso de Scrum

4.2 Planificación inicial

Para la planificación de las tareas a lo largo del tiempo, se va a detallar una tabla en la que figuren cada una de las etapas de desarrollo, y otra en la que se detallen las tareas de la etapa de implementación y pruebas. Para que la tabla no se haga excesivamente larga, se ha decidido agrupar los requisitos en un menor número de tareas, siempre y cuando los requisitos sean semejantes. Para las estimaciones vamos a usar como medida de tiempo las horas de trabajo de una sola persona.

4.2.1 Estimación y planificación de las tareas

Las tareas del desarrollo completo se detallan en la siguiente tabla:

Tarea	Descripción	Clave	Subtareas
Evaluación	Evaluación del sistema que se quiere desarrollar	BH-1	
Análisis	Análisis del sistema a desarrollar	BH-2	BH-3, BH-4
	Obtención de los requisitos	BH-3	
	Detalle de los casos de uso	BH-4	
Gestión	Definir la gestión del proceso de desarrollo	BH-5	BH-6, BH-7, BH-8
	Decidir los pasos para la gestión de las tareas del proyecto	BH-6	
	Redactar la planificación inicial	BH-7	
	Especificar el presupuesto	BH-8	
Diseño	Especificar el diseño de la aplicación	BH-9	BH-10, BH-11, BH-12
	Definir la arquitectura de la aplicación	BH-10	
	Modelar el diseño del modelo de datos	BH-11	
	Diseñar la interfaz	BH-12	
Implementación + Pruebas	Se detalla en la siguiente tabla	BH-13	

Tabla 64. Tareas del desarrollo del proyecto

De los requisitos extraídos de la fase de análisis vamos a obtener la siguiente lista de tareas a completar para terminar la etapa de implementación y pruebas. Como se ha comentado anteriormente, se ha decidido agrupar los requisitos que tengan cosas en común, para hacer más legibles y trazables las tareas que hay que realizar. Para ser más concreto, vamos a utilizar el verbo gestionar para englobar a crear, modificar, eliminar y listar elementos, de tal forma que la gestión de usuarios se refiera a la creación, edición, borrado y listado de usuarios. La lista completa es la siguiente:

Tarea	Descripción	Clave	Subtareas	Requisitos
Preparar entorno	Instalar software necesario para el desarrollo	BH-D1		
Gestión de usuarios	Permitir al administrador gestionar usuarios y su acceso	BH-D2		F-1, F-2, F-3, F-4, F-26, F-30
Integración con JIRA	Tareas relacionadas con la integración de la aplicación con JIRA	BH-D3	BH-D4, BH-D5, BH-D6	F-11, F-12, F-13, F-14, F-15, F-19
	Configurar parámetros de integración de JIRA	BH-D4		F-11, F-12, F-13, F-14
	Obtener periódicamente los proyectos de JIRA	BH-D5		F-15
	Obtener periódicamente los tiempos imputados en cada tarea	BH-D6		F-19
Integración con Factura Directa	Tareas relacionadas con la integración de la aplicación con Factura Directa	BH-D7	BH-D8, BH-D9	F-5, F-6, F-7, F-8, F-9
	Configurar parámetros de integración de Factura Directa	BH-D8		F-5, F-6, F-7, F-8
	Obtener periódicamente los datos de los clientes de Factura Directa	BH-D9		F-9
Asociar clientes con sus datos y proyectos	Relacionar los datos obtenidos de Factura Directa con los usuarios de la aplicación y los proyectos de JIRA	BH-D10		F-10, F-16, F-17, F-18
Gestión de la bolsa de horas	Permitir a los usuarios mirar su bolsa de horas y a los administradores gestionarlas	BH-D11		F-20, F-21, F-22, F-23, F-24
Enviar correo cuando la bolsa baje de un límite	Enviar un aviso al cliente cuando su bolsa de hora este por debajo de un umbral	BH-D12		F-25
Estadísticas	Permitir al usuario ver las estadísticas de los movimientos de sus	BH-D13		F-27, F-28

	bolsas			
Listado de precios	Permitir al usuario que vea los precios de las diferentes bolsas de horas	BH-D14		F-29

Tabla 65. Tareas de implementación + pruebas

De la anterior tabla, ya podemos realizar una estimación de las horas que se van a dedicar a cada tarea, fijando una fecha de inicio y una fecha de fin estimada. Esto nos permitirá, además de poder generar un presupuesto acorde con el producto desarrollado, evaluar el avance de nuestro proyecto en cualquier instante y actuar en consecuencia si los periodos se están incumpliendo.

Por lo tanto, la estimación y planificación de tareas queda de la siguiente manera (las tareas en negrita indican que tiene subtareas y que, por lo tanto, su estimación es la suma de la estimación de las subtareas):

Clave	Título	Fecha inicio	Fecha fin	Estimación (horas)
BH-1	**Evaluación**	13/02/12	20/02/12	8 h
BH-2	**Análisis**	20/02/12	28/02/12	8 h
BH-3	Requisitos	20/02/12	26/02/12	6 h
BH-4	Casos de uso	27/02/12	28/02/12	2 h
BH-5	**Gestión**	29/02/12	15/03/12	16 h
BH-6	Gestión de tareas	29/02/12	07/03/12	8 h
BH-7	Planificación	08/03/12	13/03/12	6 h
BH-8	Presupuesto	14/03/12	15/03/12	2 h
BH-9	**Diseño**	16/03/12	01/04/12	16 h
BH-10	Arquitectura	16/03/12	19/03/12	4 h
BH-11	Modelo de datos	20/03/12	27/03/12	8 h
BH-12	Interfaz	28/03/12	31/03/12	4 h
BH-13	**Implementación + Pruebas**	31/03/12	07/05/12	184 h
	Total	13/02/12	07/05/12	232 h

Tabla 66. Planificación y estimación del proyecto

Para las tareas de la parte de Implementación + pruebas, tenemos la siguiente planificación y estimación:

Clave	Título	Fecha inicio	Fecha fin	Estimación (horas)
BH-D1	**Preparar entorno**	31/03/12	01/04/12	8 h
BH-D2	**Gestión de usuarios**	01/04/12	11/04/12	48 h
BH-D3	**Integración con JIRA**	16/04/12	19/04/12	24 h
BH-D4	Parámetros	16/04/12	17/04/12	8 h
BH-D5	Proyectos	17/04/12	18/04/12	8 h
BH-D6	Tiempos / tarea	18/04/12	19/04/12	8 h
BH-D7	**Integración con Factura Directa**	19/04/12	23/04/12	16 h
BH-D8	Parámetros	19/04/12	19/04/12	6 h
BH-D9	Datos de clientes	19/04/12	20/04/12	10 h
BH-D10	**Asociar clientes con sus datos y proyectos**	20/04/12	24/04/12	24 h
BH-D11	**Gestión de la bolsa de horas**	24/04/12	03/05/12	32 h
BH-D12	**Enviar correo cuando la bolsa baje de un límite**	03/05/12	04/05/12	6 h
BH-D14	**Estadísticas**	04/05/12	07/05/12	24 h
BH-D14	**Listado de precios**	07/05/12	08/05/12	2 h

Tabla 67. Planificación de implementación + pruebas

4.2.2 Diagramas de Gant

A continuación se ilustra, de manera gráfica, la planificación de las diferentes etapas del proyecto, y como se distribuyen a lo largo del tiempo. De esta forma, se obtiene la información más fácilmente y permite conocer el estado actual del proyecto, y si este lleva o no retrasos.

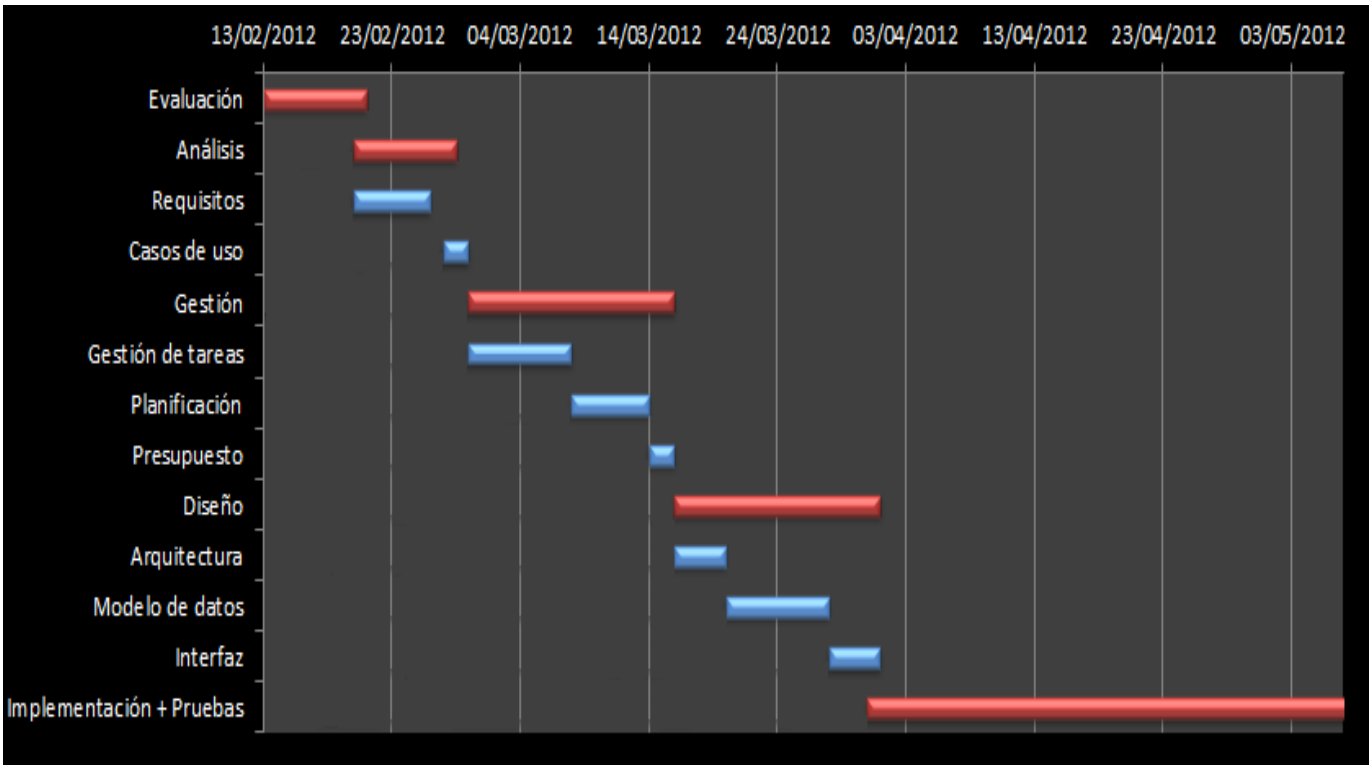

Ilustración 22. Diagrama de Gantt

4.3 Presupuesto

En este apartado se detalla el presupuesto, para el desarrollo de la aplicación, detallando los diferentes costes que implican el proceso:

PRESUPUESTO DE PROYECTO

1.- Autor:

Alberto De Ávila Hernández

2.- Departamento:

Servicio de Informática - Área de desarrollo

3.- Descripción del Proyecto:

- Titulo	**Aplicación web para la gestión de una bolsa de horas**
- Duración (meses)	**3**
Tasa de costes Indirectos:	**20%**

4.- Presupuesto total del Proyecto (valores en Euros):

6.237,39 Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a titulo informativo)	Categoría	Dedicación (hombres mes) [a)]	Coste hombre mes	Coste (Euro)
Sánchez-Mariscal, Álvaro		Ingeniero Senior	0,2	4.289,54	857,91
Ávila Hernández, Alberto de		Ingeniero	1,77	2.694,39	4.769,07
		Hombres mes	**1,97**	**Total**	5.626,98

[a)] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d)]
Equipos personales	1.600,00	100	4	60	106,67
				Total	106,67

[d)] Fórmula de cálculo de la Amortización:

$\frac{A}{B} xCxD$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado

B = periodo de depreciación (60 meses)

C = coste del equipo (sin IVA)

D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e)]

Descripción	Empresa	Costes imputable
Licencia starter JIRA	Atlassian	10,00
Licencia starter GreenHopper	Atlassian	10,00
Licencia starter Confluence	Atlassian	10,00
	Total	30,00

[e)] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,…

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	5.627
Amortización	107
Subcontratación de tareas	0
Costes de funcionamiento	30
Costes Indirectos	1.040
Total	6.916

Las horas de dedicación de Álvaro Sánchez-Mariscal han sido empleadas en la supervisión del proyecto y en la resolución de dudas y problemas encontrados en el transcurso del desarrollo.

4.4 Planificación final

A continuación se van a detallar los tiempos dedicados a cada tarea, comparándolos con la estimación inicial, lo que nos permitirá conocer si hemos sobrepasado las horas que se habían estimado para la duración del proyecto, si se han cumplido las fechas acordadas y en que fases hemos tenido mayores problemas, de cara a corregir errores en futuros proyectos.

4.4.1 Tiempo real de las tareas

En las siguientes tablas podemos ver las fechas en las que se realizaron las tareas, y que tiempo se ha empleado en cada una de ellas:

Clave	Título	Fecha inicio	Fecha fin	Tiempo empleado (horas)
BH-1	**Evaluación**	08/02/12	14/02/12	15 h
BH-2	**Análisis**	14/02/12	06/03/12	16 h
BH-3	Requisitos	14/02/12	25/02/12	12 h
BH-4	Casos de uso	26/02/12	06/03/12	4 h
BH-5	**Gestión**	06/03/12	12/03/12	18 h
BH-6	Gestión de tareas	06/03/12	09/03/12	5 h
BH-7	Planificación	10/03/12	11/03/12	9 h
BH-8	Presupuesto	11/03/12	12/03/12	4 h
BH-9	**Diseño**	12/03/12	27/03/12	23 h
BH-10	Arquitectura	12/03/12	14/03/12	5 h
BH-11	Modelo de datos	15/03/12	23/03/12	13 h
BH-12	Interfaz	24/03/12	27/03/12	5 h
BH-13	**Implementación + Pruebas**	27/03/12	14/05/12	171 h
	Total	08/02/12	14/05/12	243 h

Tabla 68. Planificación y estimación final del proyecto

Para las tareas de la parte de Implementación + pruebas se ha empleado el tiempo detallado a continuación:

Clave	Título	Fecha inicio	Fecha fin	Tiempo empleado (horas)
BH-D1	**Preparar entorno**	27/03/12	28/03/12	6 h
BH-D2	**Gestión de usuarios**	29/03/12	10/04/12	33 h
BH-D3	**Integración con JIRA**	11/04/12	19/04/12	35 h
BH-D4	Parámetros	11/04/12	12/04/12	7 h
BH-D5	Proyectos	13/04/12	16/04/12	15 h
BH-D6	Tiempos / tarea	17/04/12	19/04/12	13 h
BH-D7	**Integración con Factura Directa**	20/04/12	25/04/12	19 h
BH-D8	Parámetros	20/04/12	21/04/12	5 h
BH-D9	Datos de clientes	21/04/12	25/04/12	14 h
BH-D10	**Asociar clientes con sus datos y proyectos**	26/04/12	31/04/12	39 h
BH-D11	**Gestión de la bolsa de horas**	03/04/12	08/05/12	19 h
BH-D12	**Enviar correo cuando la bolsa baje de un límite**	08/05/12	08/05/12	2 h
BH-D14	**Estadísticas**	09/05/12	14/05/12	17 h
BH-D14	**Listado de precios**	14/05/12	14/05/12	1 h

Tabla 69. Planificación final de la implementación + pruebas

Como podemos ver, las tareas que más tiempo nos han llevado han sido la gestión de usuarios, puesto que al tratarse de la primera herramienta, esta tarea también conlleva otras actividades como la obtención de *plugins* y librerías o el desarrollo del aspecto visual de la aplicación. Además, tareas como las de integración han representado un coste de horas tan alto, debido a la dificultad de obtener los datos y extraer la información requerida. Por último, la asociación de clientes con sus datos y sus proyectos ha requerido más tiempo, por el uso del *drag and drop*, que aporta un gran valor al aspecto visual, pero no resulta tarea fácil integrarlo y hacerlo funcionar correctamente.

4.4.2 Diagrama de Gant

En la siguiente imagen podemos apreciar como quedan los datos mencionados en el apartado anterior representados en un diagrama de Gant:

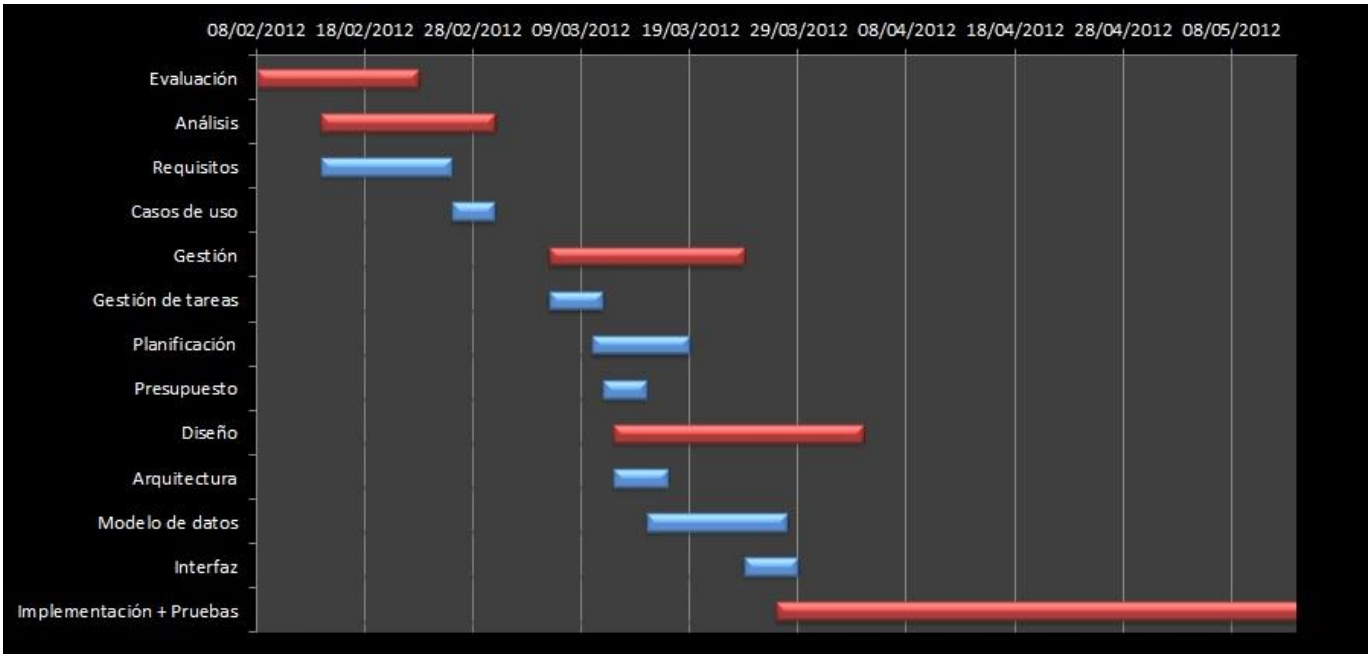

Ilustración 23. Diagrama de Gant de la planificación final

4.5 Planificación inicial vs planificación final

Para analizar el éxito del desarrollo del proyecto, debemos comparar si el tiempo inicialmente estimado ha sido menor o mayor en comparación con el tiempo real dedicado, y así saber si se han obtenido beneficios.

En primer lugar debemos comparar las fechas de las tareas, para saber si el proyecto fue entregado a tiempo, y si no fue así, en que etapas se produjeron los retrasos y porqué. En la siguiente tabla se muestra una comparativa entre las fechas iniciales y finales:

Título	Fecha inicio estimada	Fecha fin estimada	Fecha inicio real	Fecha fin real	Diferencia
Evaluación	13/02/12	20/02/12	08/02/12	14/02/12	-1
Análisis	20/02/12	28/02/12	14/02/12	06/03/12	+15
Requisitos	20/02/12	26/02/12	14/02/12	25/02/12	
Casos de uso	27/02/12	28/02/12	26/02/12	06/03/12	
Gestión	29/02/12	15/03/12	06/03/12	12/03/12	-9
Gestión de tareas	29/02/12	07/03/12	06/03/12	09/03/12	
Planificación	08/03/12	13/03/12	10/03/12	11/03/12	
Presupuesto	14/03/12	15/03/12	11/03/12	12/03/12	
Diseño	16/03/12	01/04/12	12/03/12	27/03/12	-1
Arquitectura	16/03/12	19/03/12	12/03/12	14/03/12	
Modelo de datos	20/03/12	27/03/12	15/03/12	23/03/12	
Interfaz	28/03/12	31/03/12	24/03/12	27/03/12	
Implementación + Pruebas	31/03/12	07/05/12	27/03/12	14/05/12	+11
Total	**13/02/12**	**07/05/12**	**08/02/12**	**14/05/12**	

Tabla 70. Comparación de fechas estimadas y reales

Como podemos ver en la anterior tabla, aunque el proyecto empezó varios días antes de lo previsto, en la columna diferencia podemos ver qué etapas nos han llevado más días de los planificados, como son el análisis y la implementación. Esto se debe básicamente a la inclusión de nuevos requisitos y problemas encontrados a la hora de comprender los mismos, lo que, como consecuencia, produjo un retraso mayor del esperado.

Aun así, la conclusión del proyecto tan solo se ha demorado una semana, como consecuencia de comenzar antes de lo previsto, por lo que se entiende que el proyecto se ha terminado en los plazos acordados.

A continuación se van a comparar las horas que se estimaron y las que se han empleado en el desarrollo. Este dato es importante, puesto que en función del total de horas se realiza el presupuesto y una vez conozcamos las horas totales dedicadas, podemos calcular si hemos obtenido beneficio y cuanto es ese importe. A continuación aparece reflejado la comparativa entre las horas de trabajo estimadas y las realizadas:

Título	Estimación (horas)	Tiempo empleado real	Diferencia
Evaluación	8 h	15 h	+7
Análisis	8 h	16 h	+8
Requisitos	6 h	12 h	+6
Casos de uso	2 h	4 h	+2
Gestión	16 h	18 h	+2
Gestión de tareas	8 h	5 h	-3
Planificación	6 h	9 h	+3
Presupuesto	2 h	4 h	+2
Diseño	16 h	23 h	+7
Arquitectura	4 h	5 h	+1
Modelo de datos	8 h	13 h	+5
Interfaz	4 h	5 h	+1
Implementación + Pruebas	184 h	171 h	-13
Total	**232 h**	**243 h**	**+11**

Tabla 71. Comparación de horas estimadas y reales

Como podemos observar en la tabla, todas las etapas del desarrollo han requerido más tiempo del previsto, pero en parte se ve compensado por la etapa de implementación + pruebas, por lo que la diferencia de horas es de 11 horas más de las estimadas inicialmente.

A la vista de estos resultados, el porcentaje de beneficio se ha visto reducido, ya que el sobrecoste que representan esas horas es de 215,55€.

Puesto que este coste representa aproximadamente un 3% del coste total del proyecto, el beneficio no se ve afectado en gran medida y por lo tanto, el desarrollo del proyecto ha sido todo un éxito.

En esa cifra de horas totales dedicadas al desarrollo del proyecto no están incluidas las horas dedicadas a la redacción del presente documento. Para esta tarea se han empleado unas 90 horas aproximadamente, dedicadas a la búsqueda de información, dar formato al documento y explicar en detalle cada uno de los apartados que contiene este trabajo.

Sección 5

5 Diseño de la aplicación

En esta sección se va a detallar el diseño de la aplicación, que contempla tanto la arquitectura de la aplicación, el diseño del modelo de datos y la interfaz gráfica que se le muestra al usuario de la aplicación.

5.1 Arquitectura de la aplicación

En este apartado se va a tratar la arquitectura de la aplicación, lo que incluye dos arquitecturas bien diferenciadas. Por un lado tenemos la arquitectura que ven los clientes, que es una arquitectura cliente servidor, típica de las aplicaciones web, y por otro lado, la arquitectura interna de la aplicación, basada en el patrón modelo vista controlador. Ambas arquitecturas se explican a continuación.

5.1.1 Arquitectura cliente servidor

Esta arquitectura está compuesta por dos elementos, como el propio nombre indica, el cliente y el servidor. Por un lado, el servidor que contiene la aplicación o el servicio a ofrecer, se encuentra a la espera de recibir peticiones y servirlas. Por otro lado el cliente es el encargado de realizar peticiones a través de una red de comunicación al servidor.

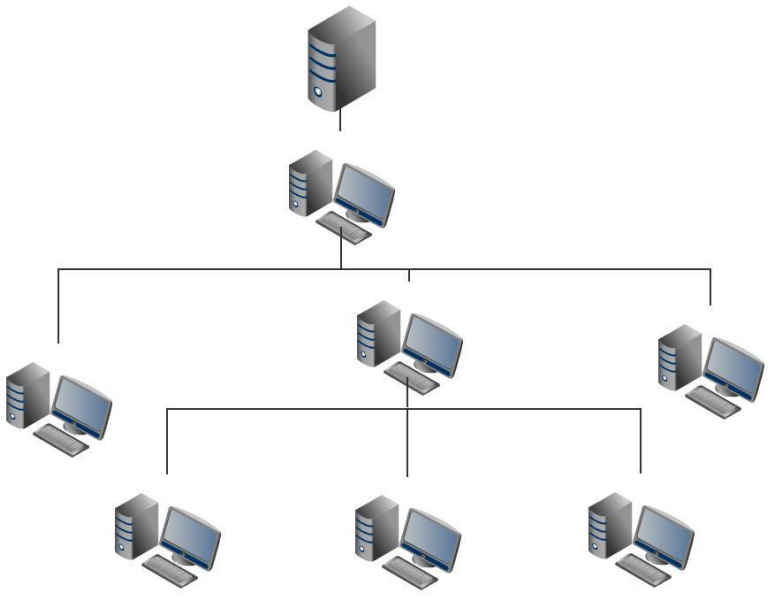

Ilustración 24. Arquitectura cliente servidor

En el caso de una aplicación web, el cliente, a través del navegador y de la conexión a internet, realiza una petición introduciendo una dirección URL. Una vez le llega al servidor la petición, este le sirve el recurso solicitado.

Para el desarrollo de esta aplicación, la información manejada va a provenir de tres fuentes diferentes, como ha quedado especificado en los requisitos. Por un lado, la información que introducen manualmente los administradores, por otro lado la información que proviene de la aplicación

JIRA y por último, de la aplicación Factura Directa, estas dos ultimas fuentes, obteniendo su información a través de REST.

Por lo tanto, la aplicación también seguirá una arquitectura cliente servidor, en referencia a que hará el papel de cliente, realizando peticiones y consumiendo información de las otras dos aplicaciones, pero a la vez también de servidor, cuando sea el usuario quien introduzca o utilice la información, como indica el siguiente diagrama:

Ilustración 25. Modelo de interacción de la aplicación

Para el intercambio de información entre Factura Directa y de JIRA con la aplicación se va a utilizar REST, con la diferencia del formato en el que vienen los mensajes:

- Para JIRA la información que recibe la aplicación viene en formato JSON.
- Para Factura Directa la información viene en formato XML.

5.1.2 Modelo vista controlador (MVC)

El modelo vista controlador es una arquitectura software que estructura un determinado sistema en tres capas. Fue descrito por primera vez en 1979

por Trygve Reenskaug [57], que trabajaba investigando en Xerox. Las tres capas de este modelo son:

- El modelo de datos: es la representación de los diferentes datos almacenados en la base de datos y los mecanismos de acceso a esta información.
- La vista: representa el conjunto de interfaces que ven los usuarios a través del navegador.
- Los controladores: son los mecanismos encargados de interactuar entre la vista y el modelo de datos.

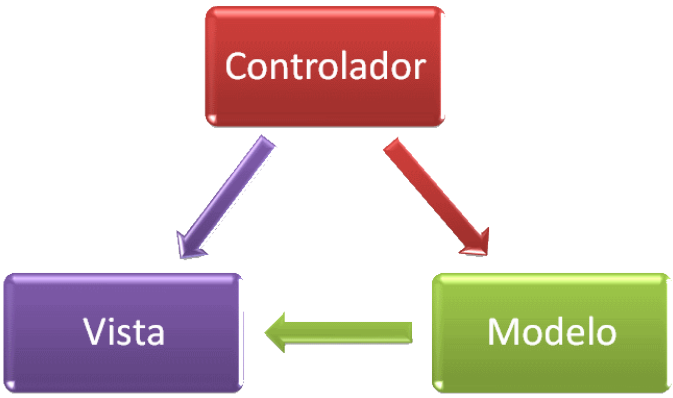

Ilustración 26. Modelo vista controlador

Cuando un usuario realiza cualquier acción sobre la vista, son los controladores los encargados de realizar la opción requerida y si corresponde, realizan consultas o modificaciones sobre los datos, aunque también puede ser que el modelo interactúe directamente con la vista sin pasar por el controlador.

Existen múltiples *frameworks* que utilizan este patrón, como por ejemplo, Cocoa [58], que emplea el lenguaje Objetive-C (utilizado por Apple para el desarrollo de aplicaciones), Ruby on Rails [59] que usa Ruby como lenguaje de programación, Struts [60], Spring [61] y JSF [62] para Java, Django [63] para Python [64], ASP .NET MVC[65] del lenguaje .NET y por último Grails, que puede ser utilizado tanto con Java como con Groovy, y es el *framework* que se utilizará para el desarrollo de la aplicación.

El modelo vista controlador tiene como principal beneficio la flexibilidad a la hora de realizar cambios, debido a la modularidad de sus tres capas, haciendo posible cambiar una de ellas, sin tener que cambiar las otras dos. Dado que el desarrollo de este proyecto puede sufrir cambios, ya sea modificando requisitos o agregando nuevos, esta arquitectura es la idónea para la construcción de la aplicación.

En las siguientes secciones se explican por separado cada una de las capas del modelo vista controlador, especificando el contenido de todas ellas.

5.2 Modelo de datos

En este apartado vamos a tratar sobre el modelo de datos que va seguir la aplicación y las diferentes tablas, con sus atributos, con las que va a contar. También se mencionan los modelos de datos de las aplicaciones con las que interactúa y de las que se extrae la mayoría de la información.

5.2.1 UML

UML son las siglas de Unified Modeling Language (Lenguaje unificado de modelado), que es un lenguaje visual que permite ver, documentar, especificar y construir sistemas. Este lenguaje esta soportado por Object Management Group [65], que es un consorcio dedicado al cuidado de varios estándares orientados a objetos, como por ejemplo, XMI y CORBA.

Con UML definiremos como se relacionan las entidades de dominio de la aplicación, y que atributos tienen estas entidades. Para ello, se utilizarán los siguientes elementos:

- Clase de dominio: describe el objeto que representa.
- Atributos: son las distintas propiedades con las que cuenta un objeto. Por ejemplo, un objeto coche, tiene atributos como peso, altura, color, etc.
- Relación: define como se relacionan diferentes clases de dominio. Las relaciones son bidireccionales pero no se leen de la misma forma en un sentido que en otro.
- Numeración de la relación: especifica cuantos objetos de una de las partes de la relación pueden interactuar con el elemento del otro extremo de la relación.

En la siguiente imagen podemos ver un ejemplo de como un objeto "coche", tiene varios atributos, como son peso y color (entre otros muchos que puede tener) y a su vez tiene objetos rueda. Un objeto rueda tiene atributos como el radio, la anchura o la presión. En la relación entre coche y rueda, podemos ver que un coche puede tener de 0 a n ruedas. Si leyéramos la relación al revés, esta se leería así: una rueda es "tenida" por un coche.

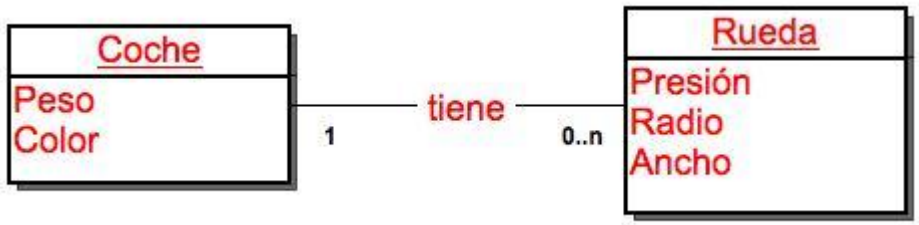

Ilustración 27. Ejemplo UML

5.2.2 Modelo relacional

En los siguientes apartados se va a explicar el modelo de datos con el que va a contar la aplicación. Además de estas clases, se va a explicar el modelo de datos tanto de JIRA como de Factura Directa, esenciales para conocer como se estructura la información que se obtiene de las llamadas REST en la aplicación y que se ven involucrados directamente en el diseño del modelo de datos.

5.2.2.1 Modelo relacional de la aplicación

En este apartado se van a especificar los diferentes objetos con los que va a contar la aplicación y como se relacionan entre ellos. En primer lugar definimos las relaciones que mantienen las clases y los números de las relaciones, para a continuación, describir cada una de las clases con sus atributos:

Ilustración 28. UML: relaciones

Las clases en detalle son:

- La clase rol: representa un tipo de usuario y sus atributos son el identificador y el nombre del rol.

Ilustración 29. UML: rol

- La clase usuario: representa a una persona que accede a la aplicación. Sus atributos son el identificador, el nombre, los apellidos, el correo

electrónico, la contraseña cifrada y el identificador del cliente al que pertenece.

Ilustración 30. UML: usuario

- La clase cliente: representa a una empresa o contiene el CIF del cliente (que a su vez hace de identificador único del cliente) y el nombre del cliente.

Ilustración 31. UML: cliente

- La clase bolsa: representa la posibilidad de que un usuario tenga varias bolsas, y en cada una de ellas disponga de diferentes proyectos asociados. Sus atributos son: el identificador, el nombre de la bolsa y el identificador del cliente al que pertenecen.

Ilustración 32. UML: bolsa

- La clase proyecto: representa un conjunto de tareas de una misma aplicación que pertenece a un cliente. Los atributos de un proyecto son el identificador, el nombre del proyecto y el identificador de la bolsa asociada a un cliente que contiene el proyecto.

Ilustración 33. UML: proyecto

- La clase movimiento: representa una tarea de un proyecto. Sus atributos son el identificador, el identificador de la tarea de JIRA, el título del movimiento, el tipo de movimiento (que indica si se trata de un movimiento hecho por el administrador de la aplicación o un movimiento automático que viene desde JIRA), el identificador del proyecto al que pertenece, el tiempo empleado en la resolución de esta tarea, la fecha en la que se finalizo la tarea y el identificador del Zendesk.

Ilustración 34. UML: movimiento

- La clase opción: representa las diferentes opciones de configuración de la aplicación. Sus atributos son: el identificador, el código (nombre del valor que representa), el valor y el tipo.

Ilustración 35. UML: opción

- La clase datos cifrados: son un tipo particular de opciones que requieren que los datos almacenados estén cifrados, por lo que se almacenan en una tabla alternativa. Los atributos que contiene esta clase son: un identificador y el dato cifrado.

Ilustración 36. UML: datos cifrados

5.2.2.2 Modelo relacional de JIRA

Con la adquisición de una licencia de la aplicación de gestión de incidencias JIRA, Atlassian ofrece su producto para que se instale en un servidor, para así poder gestionarlo de forma autónoma. Esto nos ofrece la posibilidad de crear una base de datos en el mismo servidor en el que se encuentra instalada y poder conocer cómo se gestiona la información en esa aplicación.

Una vez se instala JIRA, se crea una base de datos con 124 tablas que almacenan toda la información necesaria para que la aplicación funcione. Durante la etapa de análisis se ha realizado un estudio de cuales son estas tablas y de que información es requerida para el desarrollo de la aplicación de gestión de bolsas de horas.

En la siguiente imagen se detallan algunas de las tablas que es necesario conocer para, a posteriori, extraer la información que proviene de REST. Estas tablas no están completas y solo contemplan la información relevante para el uso que queremos hacer de esa información, que son la tabla tarea y algunas tablas con las que guarda relación:

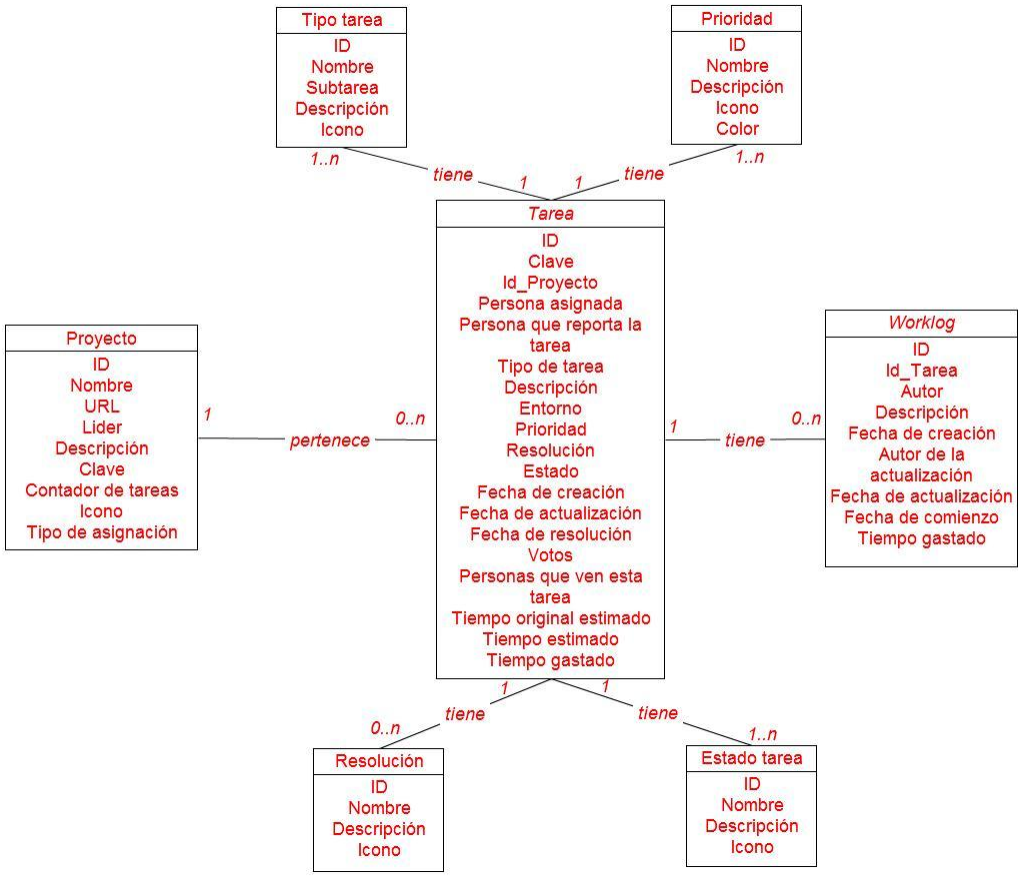

Ilustración 37. Modelo de datos parcial de JIRA

A continuación se va a explicar en detalle cada una de las tablas. Todas ellas tienen en común un identificador único numérico y autoincremental:

- La tabla proyecto: contiene información sobre un proyecto, como el nombre, el líder del proyecto, la descripción del mismo, una clave identificativa, el número de tareas que tiene, un icono y una estrategia de asignación de tareas de ese proyecto. Un proyecto puede tener de 0 a n tareas.
- La tabla tarea: es la tabla más extensa y sobre la que se centra la aplicación JIRA. Es la representación de un problema correspondiente a un proyecto. En esta tabla podemos encontrar el nombre de la tarea y su descripción, el proyecto al que pertenece, la persona asignada a esta tarea y la persona que la ha reportado, el tipo de tarea, la prioridad, el estado y el tipo de la tarea (que explicaremos a continuación), fechas sobre la tarea, quién puede ver la tarea, y tiempos de la tarea (el tiempo estimado original, el tiempo estimado actual y el tiempo empleado.
- La tabla *worklog*: representa un gasto de tiempo en una tarea, y contiene información como la fecha, el tiempo gastado, la descripción, el autor, la fecha de actualización y el autor de la actualización. Una tarea puede tener de 0 a n *worklog* puesto que hay tareas que nunca se realizan o que resulta no ser una tarea.
- La tabla resolución: indica el resultado de la tarea, con atributos como el nombre, la descripción o el icono de la resolución. Pueden existir varias resoluciones porque una tarea se puede resolver una vez, y volverse a abrir porque no se ha solucionado totalmente.
- La tabla estado tarea: representa el estado en el que se encuentra una tarea, y contiene el nombre, la descripción y el icono. JIRA permite configurar la máquina de estados que viene por defecto o crear la nuestra propia. Por lo tanto, una tarea puede transitar por varios estados.
- La tabla tipo tarea: representa si la tarea trata de un error, una nueva funcionalidad, u otros tipos que contempla la aplicación. Además de los tipos de la aplicación se pueden agregar nuevos. Una tarea puede cambiar varias veces de tipo.
- La tabla prioridad: representa la importancia que tiene la tarea y contiene información sobre la descripción de la prioridad, el icono, el color a mostrar en JIRA y el nombre. JIRA permite también crear nuevas prioridades y hacer uso de ellas. Una tarea, al igual que en los anteriores caso, puede cambiar de prioridad varias veces.

5.2.2.3 Modelo relacional de Factura Directa

La aplicación Factura Directa es una aplicación web alojada en un servidor, a la que, por el pago de una suscripción, se concede acceso a una serie de funcionalidades. Es por esta razón por la que no conocemos qué modelo de datos tiene la aplicación, aunque mediante ingeniería inversa se puede reconstruir un modelo de datos simplificado de la parte que nos interesa de la aplicación.

Del proceso de ingeniería inversa se ha extraído el siguiente modelo:

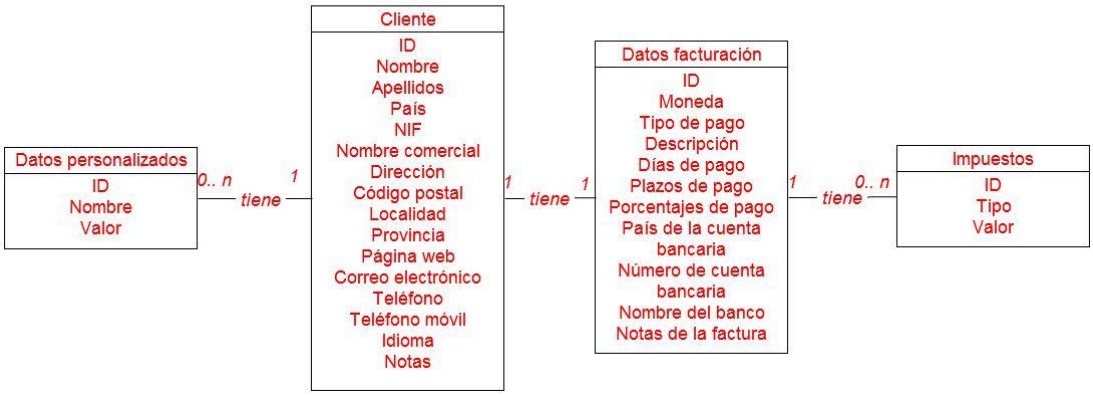

Ilustración 38. Modelo de datos parcial de Factura Directa

El esquema anterior que va a interactuar con nuestra aplicación mediante REST, contiene las siguientes tablas:

- La tabla cliente: representa una persona física o jurídica y los datos relevantes para la generación de facturas. Entre estos datos podemos encontrar información sobre el nombre, NIF, dirección y otra información relevante.
- La tabla datos personalizados: representa la posibilidad de agregar algún dato más al cliente, aunque no este entre los que vienen por defecto. Contiene los atributos de nombre y el valor correspondiente a ese campo.
- La tabla datos de facturación: es la representación de la información necesaria para generar la factura y gestionar los pagos. Contiene la información referente a los pagos, las cuentas bancarias a la que hacer los cargos, etc.
- La tabla impuestos: representa los diferentes tipos de impuestos que pueden tener los datos de facturación. Sus atributos, similares a los de la tabla de datos personalizados, son el tipo y el valor.

Una vez concluido el proceso de ingeniería inversa, descubrí en Factura Directa una opción que permitía exportar los datos en formato XML, lo que facilitaría la construcción del modelo de datos. Una vez comparado con el modelo extraído de la ingeniería inversa y no observar ningún cambio destacable, se mantiene el diseño inicial.

5.2.3 Esquema de los datos

Para la creación de la base de datos se va a implementar el modelo mencionado en el apartado anterior y los tipos de datos seleccionados para los

atributos se especifican en el siguiente esquema (con la introducción de una tabla intermedia que relaciones usuarios con su rol, puesto que un usuario puede tener varios roles):

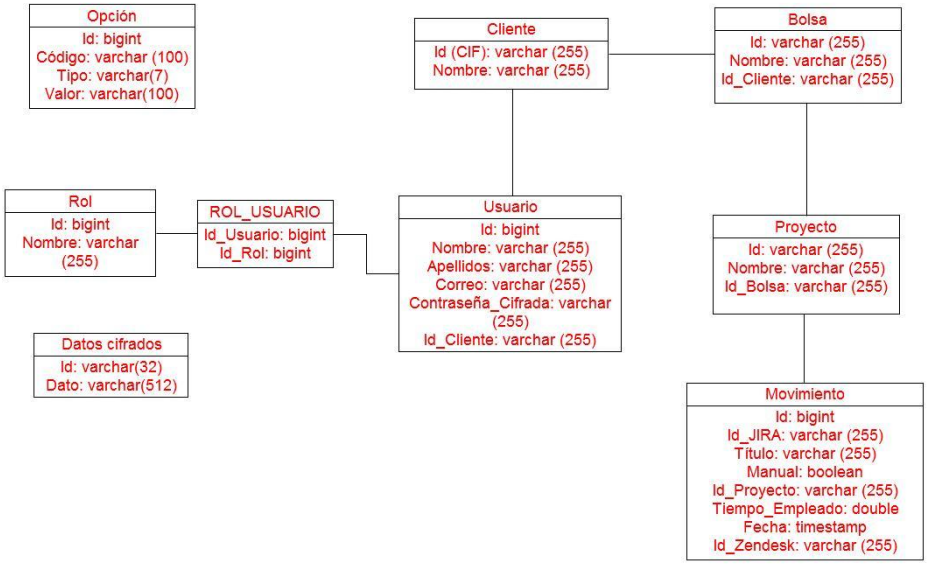

Ilustración 39. Esquema de la base de datos

Para los identificadores de las tablas que son del tipo *bigint*, serán números únicos y autoincrementales. Para los que son *varchar* únicamente se garantizara que la cadena sea única.

En el caso de las claves ajenas, se configurará la base de datos para garantizar que no existan inconsistencias, y que cuando se elimine una fila de una tabla, no existan otras filas que hagan referencia a la borrada.

5.2.4 Modelo de clases

Las clases descritas en el modelo de datos de la aplicación y que serán desarrolladas se detallan a continuación, especificando los atributos que contendrán y los métodos particulares de cada clase:

- La clase *bolsa*: representa una agrupación de movimientos, que pertenece a un cliente. Esta clase tiene:
    - Un atributo nombre de la bolsa.
    - Un atributo cliente al que pertenece esta bolsa.
    - Una lista de proyectos que están asignados a esa bolsa.
- La clase *cliente*: representa los datos como empresa. Contiene:
    - El atributo CIF, que es el código de identificación fiscal.
    - El nombre de la empresa.

- Una lista de bolsas que posee.

Además contiene un método que devuelve el nombre y el CIF separado por un guion.

- La clase *datos cifrados*: pertenece al *plugin* Crypto y se utiliza para almacenar en la base de datos las contraseñas de las aplicaciones JIRA y Factura Directa. Cuenta con varios métodos para cambiar la contraseña y sus atributos son:

    - Nombre del campo encriptado
    - Valor a guardar cifrado.

- La clase *movimiento*: representa un aumento o disminución del saldo de una bolsa. Tiene los siguientes atributos:

    - La bolsa a la que pertenece.
    - El título del movimiento.
    - El tiempo empleado.
    - El tipo de tarea.
    - La fecha de actualización de la tarea.
    - El identificador de la tarea de JIRA.
    - El identificador del ticket de Zendesk asociado a esa tarea.
    - El proyecto al que pertenece esta tarea. Este campo no es redundante con el de la bolsa, puesto que algunos movimientos no están asociados a un proyecto.
    - Una variable que indica si el movimiento ha sido introducido a mano.

- La clase *proyecto*: representa un producto o servicio ofrecido al cliente. Contiene:

    - La clave del proyecto, que proviene de JIRA.
    - El nombre del proyecto.
    - La bolsa a la que pertenece.
    - Una lista de movimientos que contiene el proyecto.

- La clase *rol*: representa los tipos de usuario que pueden existir en la aplicación, de tal forma que se puedan extender sin hacer grandes cambios en la aplicación. Sus atributos son:

    - Nombre del rol.
    - Tipo de autorización del rol.

- La clase *usuario*: es la representación de la cuenta de acceso de una persona. Entre sus atributos están:

    - Nombre de usuario, que es el correo electrónico.
    - La contraseña, que se almacena cifrada.
    - El nombre.

    - Los apellidos.
    - El cliente al que pertenece este usuario.

  Además de lo anterior, cuenta con varios métodos:

    - Un método para obtener los roles del usuario.
    - Un método para cifrar la contraseña cuando se crea un nuevo objeto usuario.

- La clase *usuario-rol*: es la clase que representa la relación entre un usuario y un rol, puesto que un mismo usuario puede tener varios roles a la vez. Por lo tanto, sus atributos son:

    - Un usuario.
    - Un rol.

  También contiene métodos para gestionar esa relación, es decir, para crear, borrar y obtener los roles de un usuario.

5.3 Vistas de la aplicación

Para el desarrollo de la aplicación, debemos especificar cuales serán las diferentes interfaces que se deberán desarrollar. Esto nos ayudará a conocer la estructura que tendrá la parte visual de la aplicación, qué elementos tiene cada interfaz y como interaccionan entre ellas.

Para clasificar mejor las interfaces, las hemos dividido en tres partes, una para cada rol de la aplicación, es decir, una para los usuarios y otra para los administradores y una para las partes comunes a los dos roles.

Los elementos que componen las interfaces diseñadas pueden ser:

- Imágenes:

Ilustración 40. Leyenda: imagen

- Texto:

Ilustración 41. Leyenda: texto

- Botones:

Ilustración 42. Leyenda: botón

- Campos para introducir texto:

Ilustración 43. Leyenda: campo de texto

5.3.1 Interfaces comunes

La única parte común que tienen el usuario y el administrador es la pantalla de acceso a la aplicación:

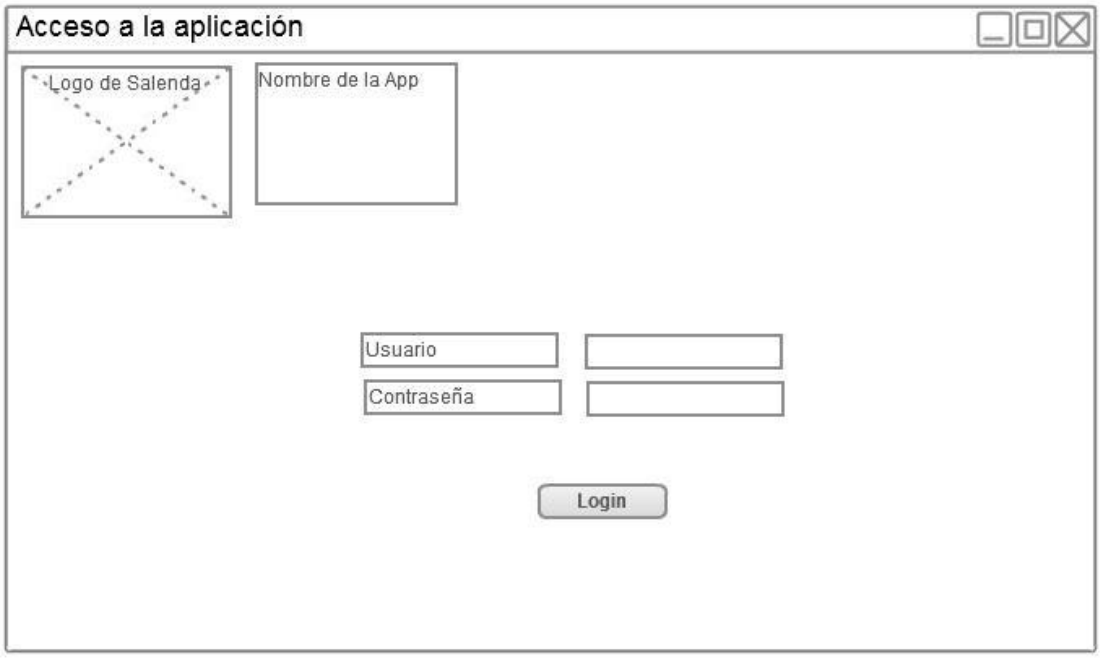

Ilustración 44. Acceso a la aplicación

5.3.2 Interfaces del usuario

El usuario tiene tres interfaces:

- Listado de bolsas y de los movimientos de esa bolsa, junto con el saldo restante. Los movimientos se cargarán una vez se haya seleccionado la bolsa:

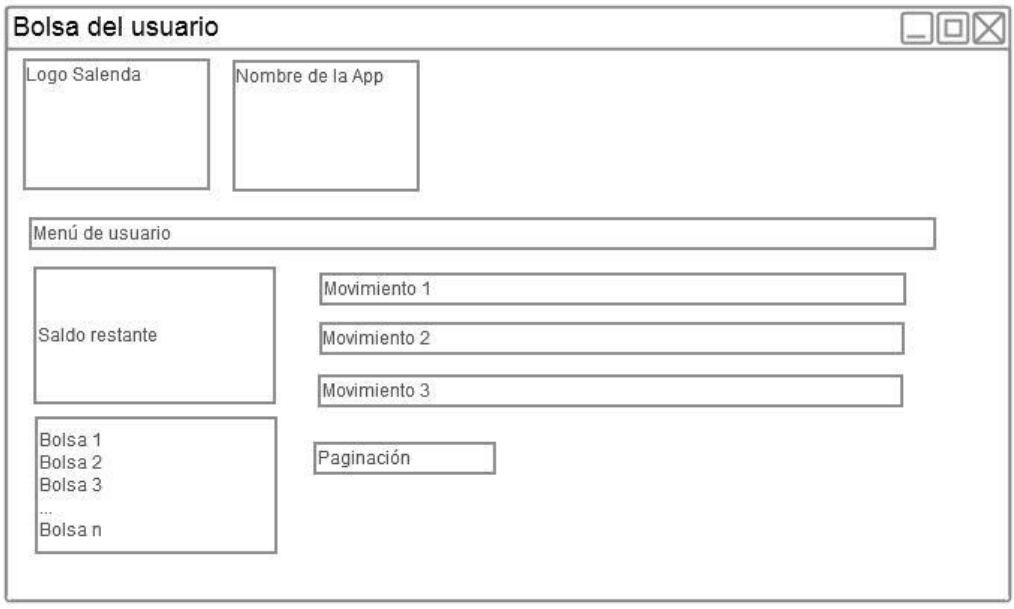

Ilustración 45. Bolsa del usuario

- Estadísticas de los movimientos:

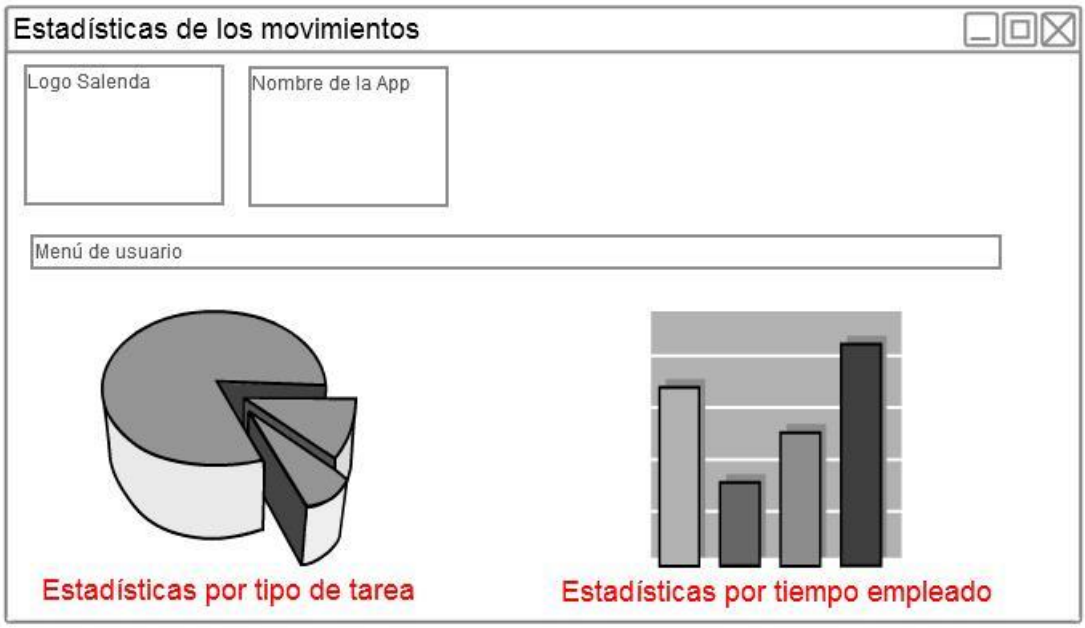

Ilustración 46. Estadísticas

- Listado de precios:

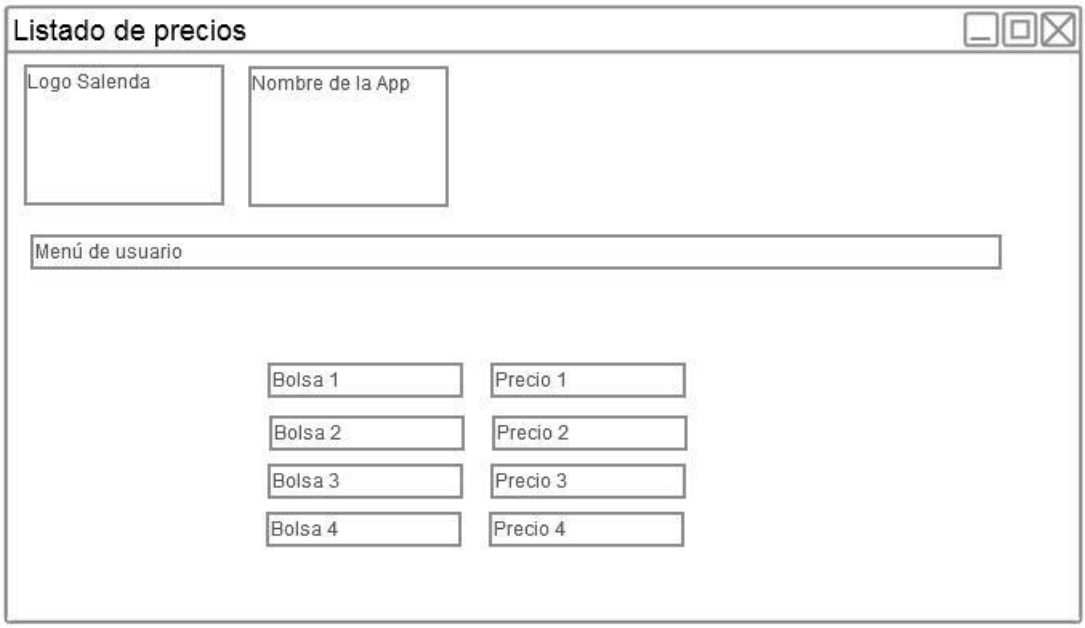

Ilustración 47. Listado de precios

5.3.3 Interfaces del administrador

El administrador cuenta con cuatro interfaces:

- Gestión de usuarios: en esta interfaz se podrá agregar, editar y modificar usuarios:

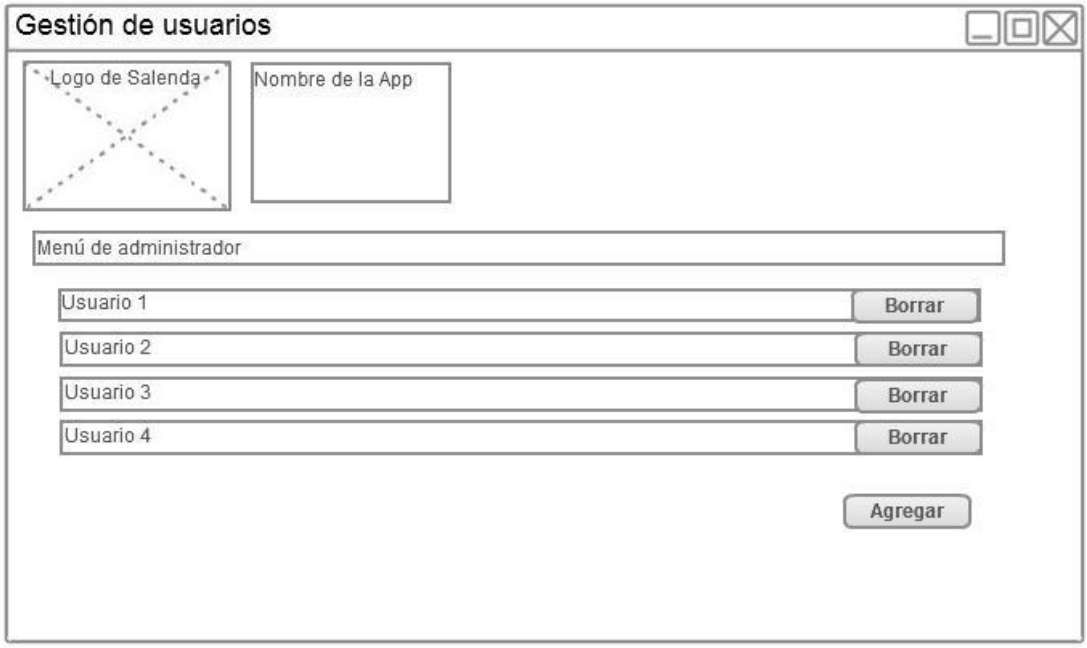

Ilustración 48. Gestión de usuarios

Para agregar usuarios, al pulsar el botón de agregar aparecerá la siguiente pantalla:

Ilustración 49. Agregar usuario

- Parámetros de integración: a través de esta interfaz se permitirán configurar los distintos parámetros para habilitar la interacción con las otras dos aplicaciones, JIRA y Factura Directa, así como los parámetros de la propia aplicación.

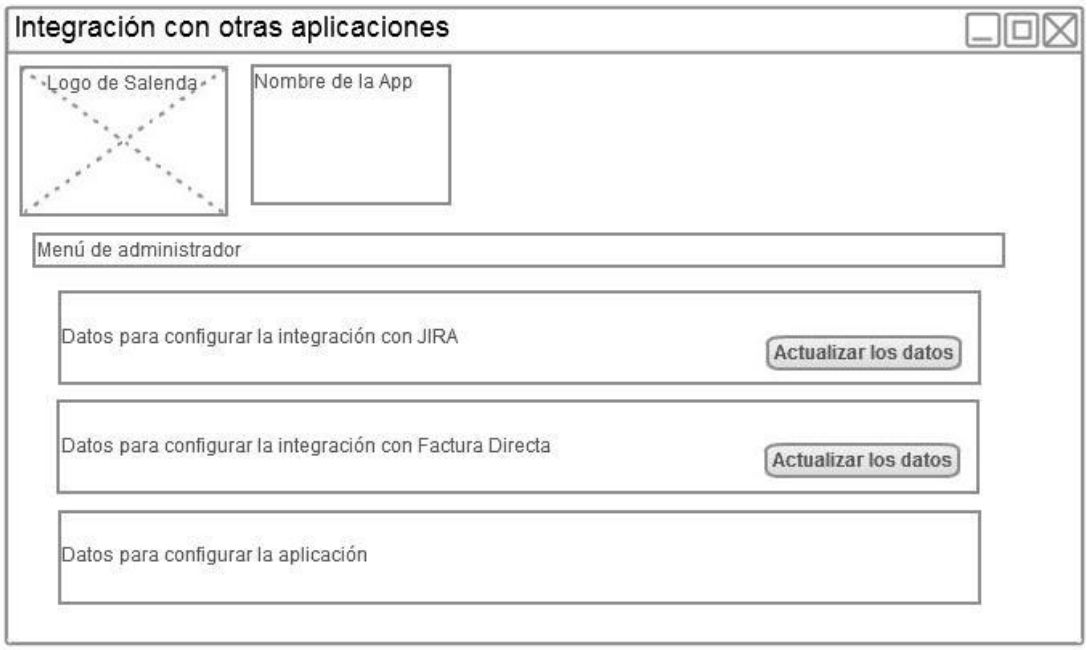

Ilustración 50. Parámetros de integración

- Asignación de proyectos: desde esta pantalla se permitirá establecer la relación entre un proyecto y la bolsa de un cliente, mediante *drag and drop*.

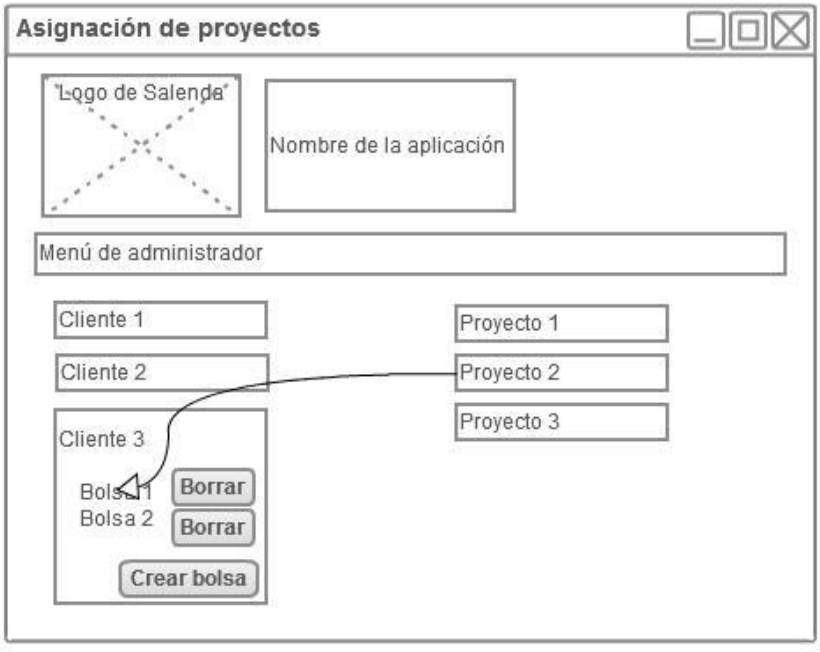

Ilustración 51. Asignación de proyectos

- Gestión de bolsas: desde esta interfaz se permite al administrador conocer el estado de las bolsas de horas de los distintos clientes, así como gestionar los movimientos, lo que incluye agregar un nuevo movimiento y reembolsar horas de un movimiento.
  Para ello, primero debemos seleccionar el cliente del que queremos ver su bolsa de horas de la siguiente interfaz:

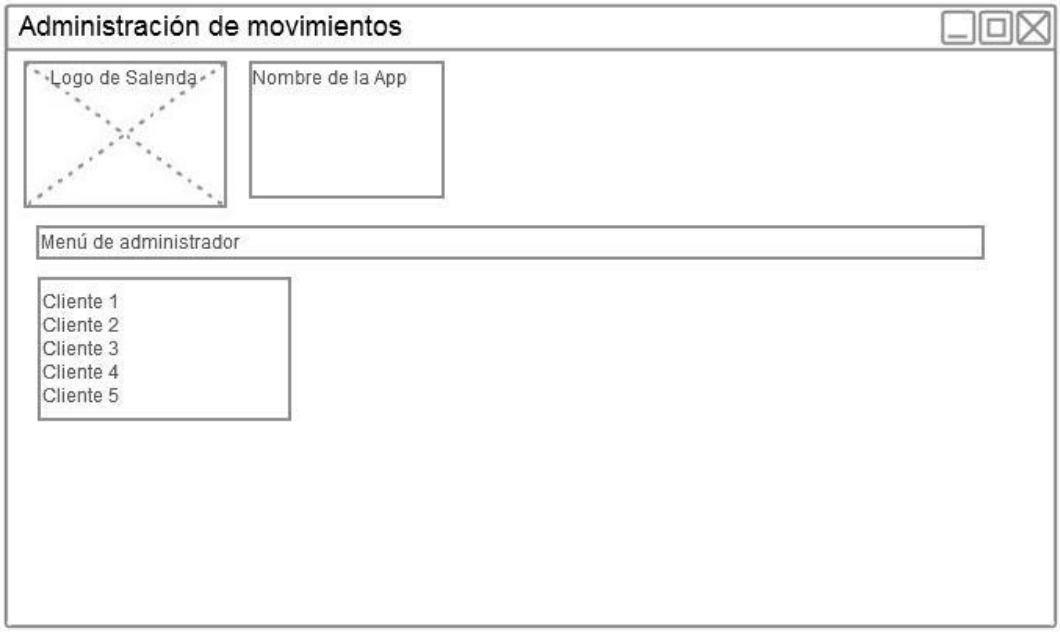

Ilustración 52. Administración de movimientos

A continuación se cargarán las bolsas de ese cliente, y seleccionando una bolsa, se cargarán los movimientos y el saldo total de la bolsa seleccionada:

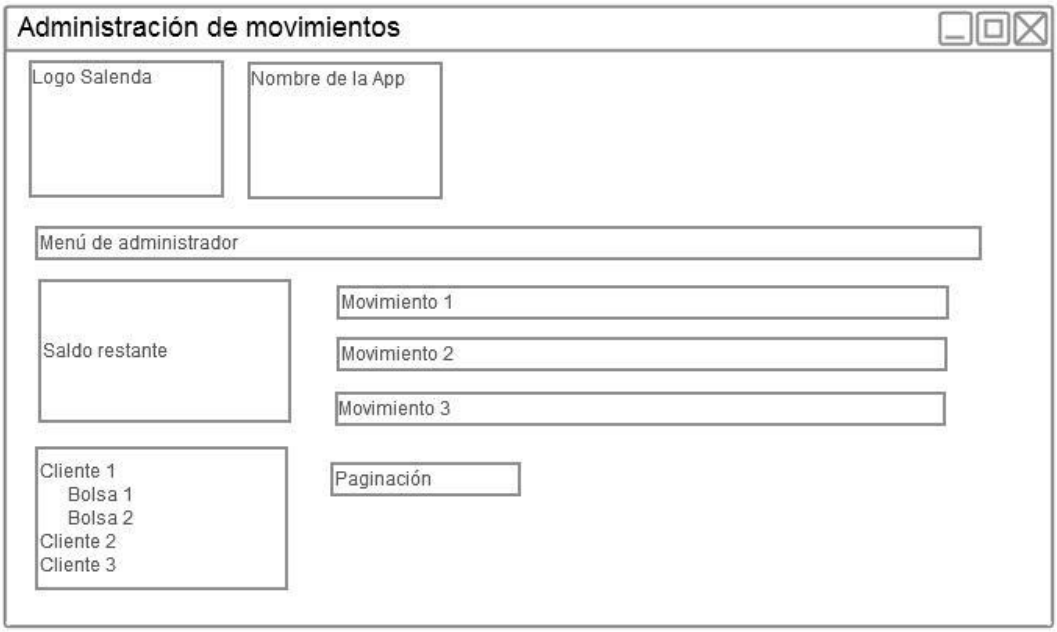

Ilustración 53. Administración de movimientos

Y desde esa interfaz podremos agregar movimientos:

Ilustración 54. Agregar movimiento

Y también reembolsar movimientos:

Ilustración 55. Reembolsar movimiento

5.3.4 Esquema global de las interfaces

En el siguiente esquema se plasma la estructura de la aplicación, detallando cada una de las secciones y que acciones se permiten realizar en cada una de ellas:

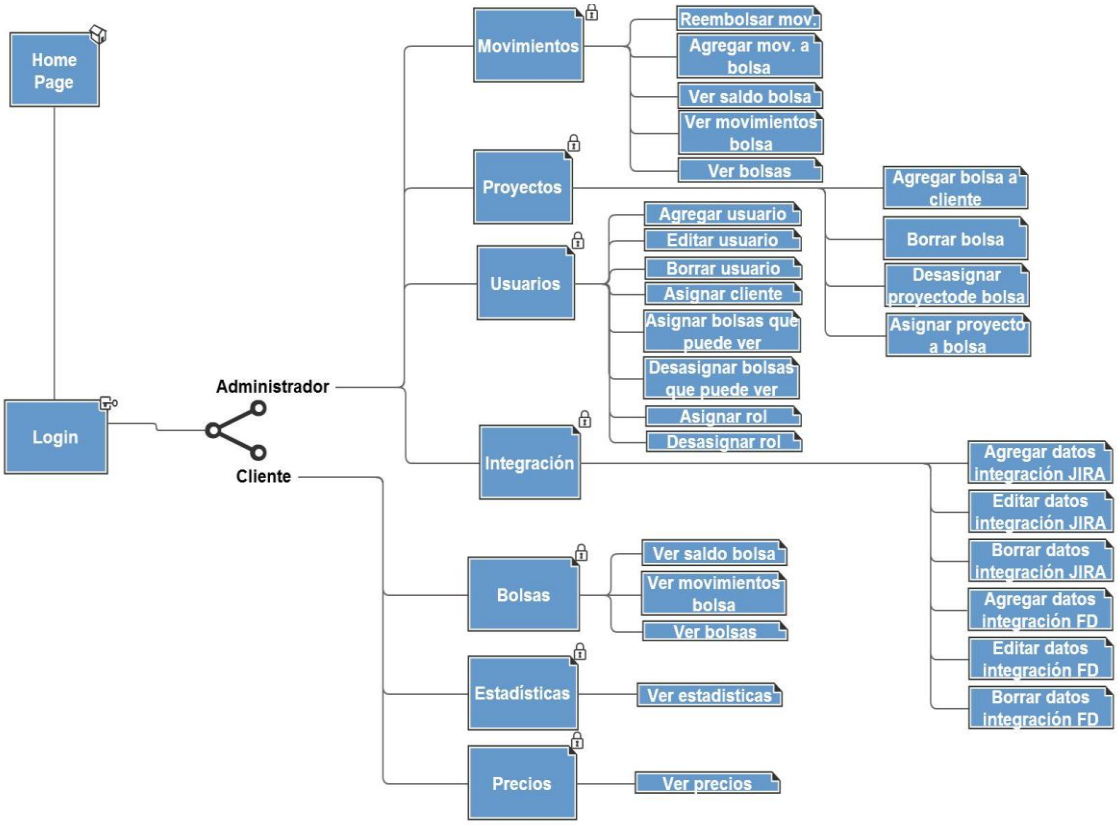

Ilustración 56. Mapa de la aplicación

5.4 Controladores de la aplicación

Para la parte de los controladores, la aplicación va a contar con uno por cada vista anteriormente mencionada:

- Controlador de gestión de usuarios: se encargará de todas las acciones que conciernen a las cuentas de usuario, entre las que están:
    - Creación de usuarios.
    - Edición de usuarios.
    - Borrado de usuarios.
    - Asignación de clientes a usuarios.
    - Asignación de roles a usuarios.

- Controlador para la integración: se encargará de la gestión de la información necesaria para la integración de la aplicación con JIRA y con Factura Directa. Las acciones que podrá realizar este controlador son:
    - Agregar información de integración.
    - Editar información de integración.
    - Eliminar información de integración.

- Controlador para la asignación de proyectos y la gestión de bolsas: permitirá todas las acciones que tienen que ver con la asignación de proyectos a bolsas, entre las que se encuentran:
    - Agregar una bolsa a un usuario.
    - Eliminar una bolsa a un usuario.
    - Asignar un proyecto a una bolsa.
    - Desasignar un proyecto de una bolsa.

- Controlador para la gestión de movimientos: permitirá al administrador realizar todas las acciones posibles desde la vista de gestión de movimientos, lo que incluye:
    - Listar los clientes.
    - Listar las bolsas de los clientes.
    - Listar los movimientos de las bolsas.
    - Agregar un nuevo movimiento.
    - Reembolsar horas a un movimiento.
    - Mostrar saldo de la bolsa.

- Controlador para mostrar las bolsas y los movimientos al usuario: similar al controlador de gestión de movimientos, anteriormente comentado, pero con un menor número de funcionalidades, que se limita a:
    - Mostrar bolsas del usuario.
    - Mostrar los movimientos de la bolsa seleccionada.

- Controlador para mostrar las estadísticas de los movimientos: este controlador realizará los cálculos sobre los movimientos, necesarios para mostrar las estadísticas. Entre estos cálculos se realizarán dos:

    - Número de incidencias por tipo: si se trata de un error, de una nueva funcionalidad, de una petición de cambio, etc.
    - Número de incidencias en función del tiempo que han gastado de la bolsa de horas.

- Controlador para mostrar la lista de precios de las bolsas de horas: este controlador solo tendrá una función, que será la redirección a la vista que muestra los precios de las bolsas de horas.

Además se incluirán varios controladores que externalizarán algunas funciones comunes a los controladores o que no tienen que ver con ninguna interfaz de usuario:

- Controlador de los correos: este controlador será responsable de dos funcionalidades principalmente:

    - Enviar un correo cuando se crea un nuevo usuario.
    - Enviar un correo cuando el saldo de una bolsa de horas esté por debajo de un límite fijado por los administradores en la interfaz de integración.

- Controlador para obtener la información de JIRA mediante REST: se encargará de realizar las dos llamadas REST y obtener los datos del resultado de las llamadas. La información que devuelven las llamadas es texto plano, por lo que hay que analizar la sintaxis y extraer la información deseada. Las llamadas solicitarán:

    - Los proyectos.
    - Los movimientos.

- Controlador para obtener la información de Factura Directa mediante REST: al igual que el anterior controlador, este se encargará de realizar la llamada REST solicitando los datos de los clientes y de la respuesta obtendrá los datos que precise la aplicación.

Sección 6

6 Implementación de la aplicación

En esta sección se va a detallar la implementación de algunas de las funcionalidades más importantes de la aplicación, como puede ser la gestión de usuarios, la configuración de los parámetros de integración, la asignación de proyectos, la obtención de los datos mediante REST, etc.

6.1 Estructura de la aplicación

Los ficheros de código se encuentran organizados en la aplicación siguiendo la estructura de carpetas que se detalla a continuación:

- Carpeta *domain*: en esta carpeta se almacenan todas aquellas clases que representan un objeto en la aplicación. En el caso de la aplicación a desarrollar, una clase de dominio puede ser la clase usuario.
- Carpeta *controllers*: en esta carpeta se sitúan los controladores del patrón MVC.
- Carpeta *views*: en esta carpeta se incluyen las vistas del patrón MVC, que serán las diferentes interfaces que podrán ver los usuarios.
- Carpeta *services*: esta carpeta contiene aquellos métodos que son utilizados o pueden ser utilizados por varios controladores, de tal forma que se externalizan y se puede acceder a ellos.
- Carpeta *plugins*: esta carpeta alberga los diferentes plugins utilizados en el proyecto.
- Carpeta *taglib*: en esta carpeta se almacenan las diferentes librerías de etiquetas propias.
- Carpeta web-app: en esta carpeta podemos encontrar las hojas de estilo (ficheros con extensión .css), las imágenes utilizadas en la aplicación y los ficheros Javascript (con extensión .js).
- Carpeta *test*: en esta carpeta se ubican los ficheros de las diferentes pruebas que se realizan en la aplicación.

En la siguiente imagen se ilustra de qué forma queda estructurado el proyecto:

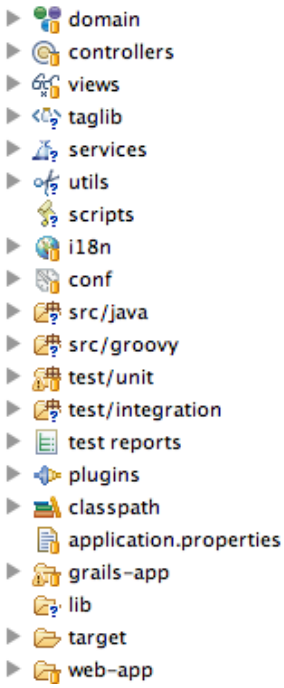

Ilustración 57. Estructura de la aplicación

6.2 Código de la aplicación

Como se ha comentado en anteriores apartados, la aplicación se estructura en carpetas, agrupando los ficheros del mismo tipo. En las siguientes secciones se van a explicar, con ejemplos, el código desarrollado que contiene cada una de esas carpetas.

En cada una de estas carpetas, los archivos están organizados dentro de un paquete denominado 'es.salenda.bolsa'.

6.2.1 Clases de dominio

Tal y como se ha comentado en el capítulo anterior, una clase de dominio se corresponde con la representación de los objetos que va a manejar la aplicación. Cada una de las clases contendrá atributos y métodos específicos de ese objeto, detallados en la anterior sección.

A continuación se va a mostrar el contenido de una de las clases de dominio representativa de todas ellas:

```
package es.salenda.bolsa


class Client {

    String cif
    String name

    static hasMany = [bags: Bag]

    static constraints = {
        cif blank: false
        name blank: false
    }

    static mapping = {
        id name: 'cif'
        version false
        id generator: 'assigned'
    }

    def getFullInfo(){
        "${name} - ${cif}"
    }

}
```

Ilustración 58. Clase de dominio Cliente

En el código anterior, podemos observar varios elementos:

- En primer lugar, el paquete donde se encuentra esta clase.
- A continuación la definición de la clase, al igual que en java. Entre el paquete y la definición de la clase irían las clases importadas si fueran necesarias.
- Seguimos con los atributos CIF, *name* y la lista de bolsas. Cuando una clase en Groovy tiene una relación con otra, dependiendo del número de objetos relacionados, podemos utilizar un atributo si se trata de una relación 1 a 1, una lista como la de este caso (representada con *hasMany*) en el lado del objeto único en una relación 1 a n, o representada con *belongsTo* si se sitúa en el otro extremo de la relación. Si se trata de una relación n a m, basta con situar a ambos lados de la relación una lista, utilizando para ello la clave *hasMany*.
- Las *constraints* representan las opciones de validación del objeto. Por defecto un atributo no puede ser nulo, pero esto se puede cambiar introduciendo el nombre del atributo, seguido de *nullable: true*. Existen también otras opciones, como puede ser longitud máxima de un texto, valor mínimo y máximo de un número, si el valor debe ser único o si se permiten valores en blanco, etc.
- El *mapping* son las opciones de configuración del mapeo entre la clase de dominio y la base de datos. En esta parte podemos especificar cual es el atributo que actuara de identificador en la tabla, si queremos que la clase tenga una versión, para conocer posibles cambios, nombres de las columnas o tablas, etc.
- Por último se especifica un método que devuelve el nombre y el CIF separados por un guion.

Los métodos de acceso a atributos de Java, comúnmente denominados *getter* y *setter* en Groovy se omiten, salvo si se desea sobreescribir la funcionalidad por defecto, en cuyo caso solo hace falta escribir el método correspondiente.

6.2.2 Controladores

Al igual que las clases de dominio, en la etapa del diseño se ha explicado la función de los controladores, como elementos que se encargan de realizar las tareas requeridas por la interacción del usuario.

Los controladores desarrollados son:

- Controlador *Home*: su único método tiene la función de redirigir a una vista u otra, una vez se accede a la aplicación haciendo *login*, en función del rol del usuario.
- Controlador de integración: se encargará de mostrar la información de integración con un método, de permitir su edición con otro y de solicitar a los servicios que actualicen los datos con otros dos métodos más. Estos servicios se explicarán en sucesivas secciones.

- Controlador de *login*: pertenece al *plugin* de SpringSecurity Core, y contiene varios métodos para garantizar el acceso a la aplicación, y a qué vista redirigir en cada caso.
- Controlador de *logout*: al igual que en el controlador anterior, pertenece al *plugin* de SpringSecurity Core y contiene un único método para cerrar la sesión del usuario en la aplicación.
- Controlador de movimientos: contiene un método para mostrar todos los clientes, una vez seleccionado uno de ellos, se utiliza otro para mostrar sus bolsas, otro para crear un movimiento, otro para mostrarlos, otro para calcular el saldo de la bolsa y por último, un método que se encarga de crear un rembolso de un movimiento.
- Controlador de precios: contiene un método que se encarga de redireccionar a la vista que contiene los precios de las bolsas.
- Controlador de proyectos: se encarga de la asignación de proyectos en bolsas y la gestión de bolsas, lo que incluye: crear y borrar bolsas (en dos métodos), listar las bolsas, los proyectos que contienen y los proyectos que no están asignados aún en otro método y un último método para asignar un proyecto a la bolsa de un cliente.
- Controlador de estadísticas: en el único método que tiene, se realizan los cálculos estadísticos de los movimientos y se envían a la vista para que, mediante Google Chart Tools, se muestren las gráficas al usuario.
- Controlador de bolsas de usuario: permite mostrar las bolsas que tiene un usuario con un método, y una vez seleccionada una bolsa, muestra los movimientos y el saldo de dicha bolsa con otro método.
- Controlador de usuarios: se encarga de gestionar los usuarios de la aplicación y hay un método por cada acción que se puede realizar sobre los usuarios, es decir, crear, editar, borrar y mostrar usuarios.

A continuación se muestra un ejemplo ilustrativo de un controlador, en este caso el controlador que muestra los movimientos a un cliente:

```groovy
package es.salenda.bolsa
import org.springframework.security.access.annotation.Secured
@Secured(['ROLE_USER'])
class UserBagController {
    def springSecurityService

    def index() {
        showBags()
    }

    def showMovements(){
        def user = springSecurityService.currentUser
        def bags = []
        def offset= (params.offset) ? params.offset.toInteger() : 0
        def movements = []
        def movementsSize = 0
        def totalBalance = 0
        def accumulatedBalance = []
        if(user.client){
            bags = Bag.findAllByClient(user.client)
            def bag = params.bag
            movements += Movement.findAllByBag(bag).reverse()
            movements.each { movement->
                totalBalance += movement.timeSpent
                accumulatedBalance.add(totalBalance / 3600)
            }
            totalBalance = totalBalance / 3600
            movementsSize = movements.size()
            int max = offset+10<movementsSize?offset+10:movements.size

            movements = movements.subList(offset, max).sort{it.updateDate}.reverse()
        }
        render (view: "listUserMovements", model: [clientSelected: user.client,bags: bags,
             movements:movements, movementCount:movementsSize,
             balance: totalBalance, offset: offset, accumulatedBalance: accumulatedBalance])
    }


    def showBags(){
        def user = springSecurityService.currentUser
        def bags = []
        if(user.client){
            bags = user.bags.sort{it.id}
        }
        render (view: "listUserMovements", model: [clientSelected: user.client, bags:bags])
    }
}
```

Ilustración 59. Controlador de los movimientos que ve un cliente

En las primeras líneas del controlador podemos encontrar lo comentado con anterioridad, el paquete donde se encuentra el fichero, y las librerías que necesita importar y la definición de la clase. Solo hay un elemento que merezca la pena que se reseñe, que se trata de la etiqueta *@Secured(['ROLE_USER'])* que se trata de una herramienta que proporciona el *plugin* SpringSecurity Core, para controlar que acciones pueden realizar los usuarios. Con esta etiqueta, se condiciona el uso de todos los métodos de este controlador a que el usuario tenga rol de usuario.

Una vez dentro de la clase, la definición de *springSecurityService* se realiza para que Grails detecte la inyección de un servicio. Esto ahorra en llamadas a objetos y permite el total acceso al servicio. Además, puesto que Groovy utiliza el tipado débil, lo que quiere decir que no hace falta especificar el tipo de un objeto, no es necesario escribir el tipo de este servicio.

Entramos con la definición de los métodos. En Groovy la política por defecto es que todos los objetos, métodos y atributos son públicos, a menos que se especifique lo contrario. De esta forma, los tres métodos *index*, *showMovements* y *showBags* devuelven un atributo, que puede ser un valor nulo, o un objeto, etc. Además estos métodos no reciben parámetros.

Una vez se declaran los métodos y estos son públicos, son accesibles desde cualquier otra vista u otro servicio o controlador que lo llame (siempre y cuando no incumpla alguna restricción de seguridad, como la de los roles mencionada anteriormente).

Dentro de cada método, se realizan las operaciones pertinentes:

- Método *index*: este método viene definido por defecto en todos los controladores, y es al que se llama cuando no se especifica el método del controlador al que se quiere llamar.
- Método *showBags*: utiliza el servicio de SpringSecurity para conocer el usuario actual y si ese usuario tiene un cliente asociado (en Groovy, nulo también significa falso, y por contraposición, un atributo cuyo valor no sea nulo, será verdadero) almacenará la lista de bolsas que posee. Por último, redirecciona a la vista *listUserMovements* pasándole un mapa compuesto por cliente del usuario en sesión y las bolsas que posee.
- Método *showMovements*: se inicializan las variables con los valores correspondientes, y se comprueba si el usuario tiene cliente, ya que si no tiene cliente asociado, no tiene bolsas con proyectos, ni movimientos que mostrar. Estos métodos pueden ser llamados desde muchos sitios, pero lo normal es que sea a través de un formulario y que este envié valores. Estos valores son accesibles a través de una variable implícita *params*, utilizado como un objeto con atributos. En este caso, *params.offset* contiene el valor de la paginación actual (y si no lo contiene, pone este valor a 0). Una vez hecho esto, se buscan todas aquellas bolsas que están asociadas a ese cliente con un *finder* dinámico. Este mecanismo permite realizar consultas sobre la base de datos sin necesidad de definir un método, gracias al mapeo de objetos relacionales. Una vez hecho esto, y mediante el mecanismo anteriormente explicado, se buscan los movimientos de esa bolsa, y se itera sobre ellos (con la operación *each*) y se realizan los cálculos para mostrar el saldo total y cómo progresa el saldo de la cuenta con cada movimiento. Por último, se calculan los diez movimientos a mostrar y se redirecciona a la vista *listUserMovements* pasando los siguientes datos:
    - El cliente.
    - Las bosas del cliente.
    - Los movimientos a mostrar.
    - El número total de movimientos.
    - El saldo de la bolsa.
    - El valor de paginación actual.
    - El valor del saldo de la bolsa después de restar cada movimiento.

6.2.3 Vistas

Las vistas son aquellos ficheros que permiten mostrar información e interactuar con los usuarios, como bien se ha explicado en el apartado del diseño.

En este caso, las vistas son páginas web con extensión GSP, que son muy similares a los ficheros JSP de Java, pero con etiquetas posibles por defecto, como puede ser el uso de sentencias condicionales, iteraciones sobre bucles o las etiquetas que sirven para dar formato a las variables mostradas, por ejemplo si se trata de un número que representa una moneda.

Un tipo especial de ficheros GSP son los *templates* que son partes que cuyo contenido se puede incluir en varias páginas GSP, de tal forma que podemos incluir este código sin necesidad de repetirlo.

La estructura que sigue un fichero GSP es la de un fichero HTML, con una parte para las cabeceras y otra para el contenido. En las cabeceras podemos definir si queremos utilizar alguna librería o recurso y le título de la página. En el cuerpo podemos encontrar mezclado código HTML junto con etiquetas de la propia GSP. Entre las más destacadas están:

- *g:if*: sentencia condicional que evalúa el contenido del parámetro *test*.
- *g:each*: forma de iterar sobre una lista, un vector o una colección de objetos. Los objetos sobre los que se va a iterar se situan en el parámetro *in*, y el objeto actual de la iteración en *var*.
- *g*:*message*: para mostrar mensajes presentes en los ficheros de internacionalización, que explicaremos más adelante.
- El uso de ${__variable__}: permite utilizar las variables que envía el controlador como modelo a la vista.

Los ficheros de vistas con los que cuenta la aplicación se estructuran en función del controlador que utilizan, de la siguiente manera:

```
<!doctype html>
<html>
    <head>
        <meta name="layout" content="main"/>
        <title> <g:message code="projects"/> </title>
        <r:require module="projects"/>
    </head>
    <body>
        <g:if test="${clients}">
            <g:form action="assign">
                <div class="width100">
                    <div class="middleRight">
                        <div id="projectsWithoutClient" class="accordion" >
                            <h3>
                                <a href="#"><g:message code="projects"/></a>
                            </h3>
                            <div class="autoHeight placeholder">
                                <g:each in="${projects}" var="project">
                                    <div id="${project.key}" class="roundedBox paddingLeft"
                                        onmouseover="cursorOver(this)" ondrag="cursorClick(this)">
                                        ${project.name}
                                    </div>
                                </g:each>
                            </div>
                        </div>
                    </div>
                    <div class="middleLeft">
                        <g:each in="${clients}" var="client">
                            <div class="projectsByClient" >
                                <h3>
                                    <a href="#">${client.name}</a>
                                </h3>
                                <div id="${client.cif}" class="autoHeight" >
                                    <g:each in="${client.bags}" var="bag">
                                        <div id="${bag.id}" class="placeholder noBackground box"
                                            onChange="this.style.color='red'">
                                            <h4 class="marginLeft marginTop">${bag.name}


                                        <g:link class="button deleteButton floatRight"
                                            action="deleteBag" params="[bag: bag.id]"
                                            title="${message(code:'deleteBag', default:'Borrar bolsa')}">
                                            <img alt="${message(code:'bag.deleteBag', default:'Borrar bolsa')}"
                                                src="${resource(file:'/images/icons/basket_delete.png')}" />
                                        </g:link>
                                        </h4>
                                        <g:each in="${bag.projects}" var="project">
                                            <div id="${project.key}" class="project paddingLeft roundedBox">
                                                ${project.name}
                                            </div>
                                            <input type="hidden" name="${bag.id}" value="${project.key}"/>
                                        </g:each>
                                    </div>
                                </g:each>
                                <div class="paddingBotton">
                                    <span id="${client.cif}" class="button"
                                        title="${message(code:'bag.add', default:'Agregar bolsa')}"
                                        onclick="showColorbox('${client.cif}')">
                                        <img alt="${message(code:'bag.add', default:'Agregar bolsa')}"
                                        src="${resource(file:'/images/icons/basket_add.png')}" /> <g:message code="bag.add"/>
                                    </span>
                                </div>
                            </div>
                        </div>
                    </g:each>
                </div>
            </div>
            <div class="floatRight clearBoth">
                <g:submitButton id="assignButton" class="button" name="assignButton"
                    value="${message(code:'projects.assign', default:'Asignar')}"/>
            </div>
        </g:form>
    </g:if>
    <g:else>
        <g:if test="${!clients && !projects}">
            <div id="warningBox" class="roundedBox">
```

```
                    <g:message code="projects.noClientsNeitherProjects"/>
                </div>
            </g:if>
            <g:else>
                <div id="warningBox" class="roundedBox">
                    <g:message code="projects.noClients"/>
                </div>
            </g:else>
        </g:else>
    </body>
</html>
```

Ilustración 60. Vista de asignación de proyectos

En este fichero podemos ver:

- Si hay clientes, muestra un formulario con dos columnas:
    - En la columna de la derecha van a aparecer los proyectos que no están asignados a ninguna bolsa.
    - En la columna de la izquierda se muestra una lista de clientes y las bolsas que tiene cada cliente. Por cada bolsa, aparecerá su nombre, un botón para borrarla y la lista de proyectos que contiene. Además, debajo de las bolsas de los clientes aparecerá un botón para agregar una bolsa a ese cliente.

- Si no hay clientes, comprueba si hay proyectos. Si los hay, muestra un mensaje, indicando que no hay ni clientes ni proyectos, y si hay proyectos, muestra un mensaje indicando que no hay clientes.

En la siguiente imagen podemos ver un ejemplo de un *template*:

```
<div>
    <h3 class="redTitle" align="center"><g:message code="bag.add"/></h3>
    <g:form name="createBagForm" action="createBag">
        <table>
            <tr>
                <td>
                    <g:message code="bag.title"/>
                </td>
                <td>
                    <g:textField name="name"/>
                </td>
            </tr>
            <tr>
                <td colspan="2" style="text-align:right">
                    <g:hiddenField name="clientCif" value="${clientCif}"/>
                    <g:submitButton id="createBagButton" class="button" name="createBagButton" value="Crear bolsa"/>
                </td>
            </tr>
        </table>
    </g:form>
</div>
<r:external uri="/js/buttons.js"/>
```

Ilustración 61. Ejemplo de *template*

Como podemos ver, un *template* solo contiene parte del código y no sigue una estructura que contenga las etiquetas *html*, *head* o *body*. En este caso, dentro de un formulario se muestra un mensaje, hay un campo para introducir texto, un campo oculto que envía el CIF del cliente y un botón para enviar al formulario. En la última línea solicitamos que se ejecute el fichero Javascript buttons.js.

6.2.4 Servicios

Un servicio es una clase que contiene funcionalidad para ser utilizada desde varios controladores y evitar así repetir código. De esta forma, un cambio en la funcionalidad del servicio solo implica el cambio en ese servicio, pero si no existieran, tendríamos código duplicado, lo que haría más costosos los cambios y una mayor dificultad en el mantenimiento de la aplicación.

En la aplicación contamos con cuatro servicios:

- El servicio de Factura Directa: este servicio cuenta con dos métodos. Uno de ellos se encarga de iniciar la conexión con los datos necesarios para que se permitan realizar peticiones REST y el otro se encarga de realizar la petición y extraer de la respuesta, obtenida en formato XML, la información que interesa para almacenarla en base de datos.
- El servicio de JIRA: al igual que el servicio de Factura Directa, este consta de un método para iniciar la conexión que posibilite las peticiones mediante REST. Además consta de tres métodos: uno de ellos se encarga de realizar la petición de los proyectos presentes en JIRA y los que no existan previamente, los almacena en base de datos, otro de ellos realiza las mismas operaciones sobre los movimientos, con la excepción de que un movimiento puede haber sido actualizado, y por último, un método que calcula si el saldo de una bolsa está por debajo del límite, en cuyo caso se enviaría una notificación al correo del usuario propietario.
- El servicio de notificaciones: servicio que se encarga de enviar los correos electrónicos. Se trata de dos correos, uno cuando se crea un nuevo usuario en la aplicación, comunicándole este hecho y cuales son los datos de acceso y otro cuando una bolsa está por debajo del límite establecido.
- El servicio de gestor de contraseñas: este servicio viene del *plugin* Crypto y contiene métodos para almacenar y obtener una contraseña almacenada en la base de datos.

En la siguiente ilustración podemos ver un ejemplo del servicio de notificaciones:

```
package es.salenda.bolsa

class NotificationService {

    def mailService
    def passwordManagerService

    def sendNewUserNotification(User user, def pass){
        mailService.sendMail {
            to user.username
            subject "Nuevo usuario en Bolsa de horas"
            body (view:"/email/newUserMail", model:[user: user, pass: pass])
         }
    }

    def sendLimitBagNotification(User user){
        mailService.sendMail {
            to user.username
            subject "Bolsa de horas de Salenda a punto de acabar"
            body (view:"/email/limitBagMail", model:[user:user])

         }
    }
}
```

Ilustración 62. Servicio de notificaciones

En el, vemos como se definen los dos métodos anteriormente mencionados y, haciendo uso del *plugin* Mail, utilizado para enviar correos, se envían al correo electrónico del usuario, indicando un asunto, y como cuerpo, una vista a la que le pasamos varios argumentos, en función del contenido del mensaje.

6.2.5 *Tareas periódicas*

Las tareas periódicas, o *jobs*, son controladores que ejecutan métodos periódicamente, siendo este momento configurable mediante expresiones cron. Las expresiones cron son cadenas de entre 6 y 7 valores que indican un momento. Esos 6 o 7 valores son los segundos, minutos, horas, días del mes, mes, día de la semana y, opcionalmente, el año. Se pueden especificar una valor numérico para cada campo o valores especiales como el valor “*” que indica todos, un rango separado por un guion, el valor “?” que indica cualquier valor, etc.

Se han creado dos tareas periódicas que se repiten todos los días a las 6 de la mañana. En una de ellas se actualizan todos los datos procedentes de JIRA, es decir, si hay nuevos proyectos, crea el objeto proyecto, si hay nuevos movimientos, los crea y si algún objeto ha sido actualizado, lo actualiza. También calcula el saldo de cada una de las bolsas, y si está por debajo del límite establecido por los administradores desde la vista de integración, se le envía un correo electrónico al usuario asociado a ese cliente. La otra tarea periódica actualiza los datos de Factura Directa, que son los clientes.

En la siguiente imagen podemos ver como es la estructura de un *job*:

```
package es.salenda.bolsa


class FacturaDirectaJob {

    def facturaDirectaService

    static triggers = {
        cron name: 'facturaDirectaJobTrigger', cronExpression: "0 0 6 * * ?"
    }

    def execute() {
        facturaDirectaService.loadClients()
    }
}
```

Ilustración 63. *Job* que actualiza la información de Factura Directa

Como se ha especificado en el apartado de los servicios, las funcionalidades de realizar la petición REST y obtener los datos están externalizadas, de tal forma que podemos acceder a ellas tanto desde los controladores, como desde los *jobs*. Por ello, este *job* solo necesita definir la dependencia del servicio, declarar la expresión cron que se encargue de lanzar el método a las 6 de la mañana cada día y dentro del método, realizar una llamada al método dentro del servicio que se encarga de actualizar los datos.

6.2.6 Javascript

El uso de Javascript en este proyecto esta orientado principalmente a mejorar el aspecto visual de la aplicación, permitiendo al usuario interaccionar con interfaces amigables, y que sean atractivas.

Existen numerosos ficheros, uno por cada vista principal de la aplicación, puesto que en cada vista, los campos tienen nombres diferentes y se requieren cosas distintas. Además de estos ficheros, existen otros ficheros que permiten aumentar las funcionalidades ofrecidas por Javascript, mediante el uso de *plugins* o de librerías.

Las principales funciones que se realizan en cada uno de los ficheros son:

- Dar un mejor aspecto visual a los botones.
- Mostrar *colorbox* (son ventanas emergentes que sombrean el contenido de la página original, para centrar la atención en la pequeña ventana que acaba de aparecer).
- Dar estilos con cajas redondeadas a varios elementos de la aplicación.
- Permitir el *drag and drop* para asignar proyectos.
- Mostrar *tooltips* (caja de texto que aparece cuando dejamos el puntero sobre una imagen o botón, explicando lo señalado) más vistosos.

En la siguiente imagen podemos ver un ejemplo de código Javascript que se utiliza en la aplicación:

```
$(document).ready( function(){
    Nifty("div.roundedBox", "big");

    $(".button").button();
    $("#addMovement").colorbox({
        href : "add?clientCif=" + $("#clientCif").val() + "&bag=" + $("#bag").val(),
        width : "25%"
    });

});

function showColorbox(id){
      $.fn.colorbox({
          width:"50%",
          href : "subtract?clientCif=" + $("#clientCif").val() + "&movementId=" + id + "&bag=" + $("#bag").val(),
      });
}
```

Ilustración 64. Javascript de la vista de administración de movimientos

En este código podemos diferenciar dos secciones:

- La primera parte contiene acciones que se van a ejecutar una vez la página se ha cargado completamente. Estas funciones son:
    - Aquellos elementos *div* que tengan como estilo la clase *roundedBox*, serán transformados en una caja con los bordes redondeados (y que, debido al estilo de esa clase, tendrá un fondo gris).
    - A los elementos que tengan como estilo la clase *button*, se transformarán en un botón con estilos de jQuery.
    - Al elemento con identificador *addMovement* (que se trata de un botón), se le ha modificado su comportamiento, de tal forma que ahora mostrará un *colorbox*, solicitado mediante la llamada al método *add* de ese contorlador, y que tendrá como ancho, el 25% del total.

- En la segunda sección podemos ver un método Javascript que recibe un identificador. Este método altera el comportamiento del elemento desde el que es llamado, mostrando un *colorbox* cuando sea llamado el evento, y la llamada que realiza este *colorbox* es al método *substract* pasando como argumentos el CIF del cliente, el identificador del movimiento y el identificador de la bolsa.

6.2.7 Librería de etiquetas

Una librería de etiquetas, o *taglib*, es un conjunto de etiquetas que se utilizan en una vista para para englobar el contenido y aplicarle unas

propiedades. Por ejemplo, las etiquetas de HTML, como por ejemplo <*div*> o <*input*>.

En este caso, hemos creado nuestra propia librería de etiquetas, que contiene una etiqueta denominada *nameUserLogged* que nos sirve para que, con tan solo escribir en las vistas <*nameUserLogged*>, la vista reconozca que es una etiqueta de mi librería de etiquetas, y ejecute el código que veis a continuación:

```
package es.salenda.bolsa

class BagTagLib {

    def springSecurityService

    def nameUserLogged = { attrs, body ->
        out << body() << springSecurityService.getCurrentUser().name
     }
}
```

Ilustración 65. Librería de etiquetas propia

Como ya hemos visto en otras ocasiones, lo que hace este código es obtener el servicio de SpringSecurity y dentro del código que se ejecuta cuando se procesa la etiqueta, devolver a la vista el nombre del usuario actual.

6.2.8 Conf

Los ficheros de configuración que tiene Grails y las funciones de cada uno de ellos, son las siguientes:

- *ApplicationResources*: define conjuntos de recursos y dependencias que se utilizan en las vistas. Si un conjunto de vistas utilizan varios ficheros Javascript y de estilos, podemos definir un conjunto que contenga esos recursos y utilizarlos en cada una de esas vistas utilizando una sola línea, ahorrando espacio y trabajo a la hora de realizar cambios.
- *BootStrap*: este fichero contiene las partes de código que se deben ejecutar al arrancar la aplicación y al destruirla. En el caso de esta aplicación, se introducen algunos valores por defecto, como un primer usuario, los roles que tendrá la aplicación y los valores por defecto que se utilizan para la integración.
- *BuildConfig*: en este fichero podemos encontrar la configuración sobre la construcción del proyecto en un fichero WAR, los repositorios para que descargue las dependencias o los *plugins* que utiliza.
- *Config*: desde este fichero se puede configurar los distintos aspectos de la aplicación, como de cada uno de sus *plugins*. Por ejemplo, en este fichero se configura, en función del entorno en el que se encuentre desplegada la aplicación, si se envían correos electrónicos o no.

- *DataSource*: en este fichero se definen las bases de datos a utilizar y la política (si se crea una base de datos nueva cada vez que arranca o si se actualiza la que ya existiera), en función del entorno en el que nos encontremos.
- *UrlMappings*: define los controladores a los que redirigir en caso de error, cuando se accede a la URL de acceso o cómo se muestra en la URL cuando se interactúa con controladores y sus métodos.

6.2.9 Otros

El resto de ficheros que han sido desarrollados son dos:

- Los ficheros de internacionalización: son ficheros clave valor que contienen los valores de los mensajes a mostrar en la aplicación. Existen múltiples ficheros, gracias a los cuales se puede traducir la aplicación de manera rápida y sencilla. En nuestro caso la aplicación solo consta de los valores para español. El aspecto del fichero es el siguiente:

```
# Banner
salenda = Salenda
app = Bolsa de horas
integration = Integración
users = Usuarios
projects = Proyectos
movements = Movimientos
userBag = Bolsa
balance = horas de saldo restante
hours = horas
logout = Salir
statistics = Estadísticas
prices = Precios
hello = Hola

# User managment
userManagment = Gestión de usuarios
userManagment.name = Nombre
userManagment.surname = Apellidos
userManagment.mail = Email
userManagment.password = Pass
userManagment.delete = Borrar
userManagment.delete.alt = Borrar usuario
```

Ilustración 66. Fichero de internacionalización

- Los ficheros de pruebas: estos ficheros serán explicados en la siguiente sección en detalle.

Sección 7

7 Pruebas

En esta sección se van a explicar las diferentes pruebas realizadas a la aplicación para comprobar que las funcionalidades implementadas funcionan correctamente de acuerdo con los requisitos expuestos en la fase de análisis.

7.1 Pruebas realizadas

La etapa de pruebas es llevada a cabo a la par que el desarrollo de la aplicación, puesto que es necesario verificar que una tarea satisface todos los requisitos antes de comenzar con la siguiente. Si una funcionalidad no pasa las pruebas, se revisa su implementación y se hacen los cambios pertinentes hasta que pasa las pruebas.

Durante el desarrollo de la aplicación, se han utilizado datos de prueba, lo que aporta seguridad a la hora de trabajar con esa información. Además, se ha utilizado una instancia de JIRA local, con datos de prueba, y se ha creado una cuenta de Factura Directa con datos ficticios. Una vez se terminó el desarrollo de la aplicación, se utilizaron datos reales para probar si todo funcionaba correctamente.

Las pruebas realizadas podemos clasificarlas en dos grupos:

- Pruebas de código unitarias: se realizan sobre funcionalidades concretas de la aplicación para verificar que funcionan correctamente. En nuestro caso, estas pruebas también podrían denominarse de integración, puesto que el objetivo es comprobar la conectividad entre la aplicación desarrollada, junto con JIRA y Factura Directa.
- Pruebas de caja negra: son pruebas realizadas proporcionando una entrada y observando la salida que esta produce.

En los siguientes apartados se profundiza en los detalles de cada una de los tipos de pruebas y a que campos han sido aplicadas.

7.2 Pruebas unitarias

Las prueba unitarias, como hemos comentado con anterioridad, pretenden verificar el correcto funcionamiento de una funcionalidad en concreto.

Las pruebas unitarias son secciones de código que se ejecutan con la aplicación parada y sin utilizar ninguna base de datos. Por eso, para probar, es necesario que en cada prueba se introduzcan aquellos datos que se van a utilizar en esa prueba.

En este caso, se han realizado pruebas unitarias a dos acciones cada una de las cuales se relaciona con un método específico:

- Obtener la información de JIRA.
- Obtener la información de Factura Directa.

7.2.1 Pruebas unitarias sobre JIRA

Para conocer si la obtención de la información que proviene de JIRA es correcta, se ha creado un test que introduce los datos necesarios para la conexión y se ha realizado una llamada para que realice la correspondiente petición de datos sobre los proyectos y sobre los movimientos. Una vez realizada la llamada, se realiza la comprobación para ver si el número de proyectos existentes en base de datos es mayor que cero, lo que verifica que la conexión funciona y que se obtiene la información adecuadamente.

```
package es.salenda.bolsa

import org.grails.plugins.settings.Setting

@TestFor(JiraService)
@Mock ([Setting, PasswordManagerService, EncryptedData, Project, Movement])
class JiraServiceTests {

    void testLoadProjects() {

        def jiraURL = new Setting()
        jiraURL.code = 'integration.jira'
        jiraURL.type = 'string'
        jiraURL.value = 'http://localhost:8090'
        jiraURL.save()

        def jiraUsername = new Setting()
        jiraUsername.code = 'integration.jiraUsername'
        jiraUsername.type = 'string'
        jiraUsername.value = 'alberto.deavila'
        jiraUsername.save()

        def passwordManagerService = new PasswordManagerService()
        passwordManagerService.store('jiraPass', '1234', 'jiraPass')

        service.loadProjects()
        assert Project.findAll().size() > 0
        service.loadWorklogs()

    }

}
```

Ilustración 67. Pruebas unitarias sobre JIRA

7.2.2 Pruebas unitarias sobre Factura Directa

El caso de las pruebas unitarias sobre Factura Directa es similar al anteriormente comentado. En esta prueba, se introducen los datos de integración con Factura Directa, y se solicitan los clientes. Para verificar que todo ha funcionado correctamente, se comprueba que el número de clientes es mayor que cero.

```
package es.salenda.bolsa

import grails.test.mixin.*
@TestFor(FacturaDirectaService)
@Mock ([Setting, PasswordManagerService, EncryptedData, Client])
class FacturaDirectaServiceTests {

    void testLoadClients() {

        def setting = new Setting()
        setting.code = "integration.facturaDirecta"
        setting.type = "string"
        setting.value = "https://bolsa.facturadirecta.com"
        setting.save()
        def setting2 = new Setting()
        setting2.code = "integration.facturaDirectaUsername"
        setting2.type = "string"
        setting2.value = "09fabd17179578742896c2248e2c1842"
        setting2.save()

        def passwordManagerService = new PasswordManagerService()
        passwordManagerService.store("facturaDirectaPass", '1234', "facturaDirectaPass")

        service.loadClients()

        assert Client.findAll().size() > 0

        service.facturaDirecta = null
        service.loadClients()
    }
}
```

Ilustración 68. Pruebas unitarias sobre Factura Directa

7.3 Pruebas de caja blanca

Estas pruebas pretenden comprobar que la funcionalidad ha sido desarrollada correctamente, para lo cual se introducen datos desde las vistas de la aplicación y se observa el comportamiento de la misma.

Para cada vista se han realizado una serie de pruebas que se detallan a continuación.

7.3.1 Pruebas de gestión de usuarios

Ilustración 69. Pruebas sobre gestión de usuarios

Para la vista mostrada en la figura 69 se han llevado a cabo las siguientes pruebas:

- Introducir un usuario con datos normales.
- Introducir un usuario sin algún dato no requerido.
- Introducir un usuario sin algún dato requerido, lo que provoca un error y no se crea el usuario.
- La creación de usuarios con la dirección de correo electrónico repetida, lo que provoca el mismo fallo anterior.
- En cada uno de los campos ha introducir se han probado varios datos, como caracteres diferentes del alfabeto, signos de puntuación, etc. Todos ellos son aceptados, salvo en el campo *email* que debe seguir el patrón de un correo electrónico.
- Borrar un usuario.
- Borrar todos los usuarios, lo que provoca que una vez cerremos la sesión, no podremos acceder a la aplicación, salvo que creemos un usuario en la base de datos.
- Se ha probado a editar todos los valores que contiene un usuario, para verificar que se cambian correctamente, y especialmente con el cliente y el rol, que son campos con valores predefinidos.

7.3.2 Pruebas de los datos de integración

Ilustración 70. Pruebas de los datos de integración

Las pruebas hechas sobre la vista de integración son las siguientes:

- Editar los valores que vienen por defecto.
- Borrar valores.
- Probar en el campo de Consulta que los caracteres “>”, “<” o “=” funcionan correctamente.
- Probar a traer los datos de JIRA y Factura Directa con la información correcta, lo que funciona correctamente.
- Probar a traer los datos de JIRA y Factura Directa con la información incorrecta, lo que muestra un error de que no se han podido obtener los datos.

7.3.3 Pruebas de la asignación de proyectos

Ilustración 71. Pruebas de asignación de proyectos

En esta vista se ha probado todo lo concerniente a la asignación de proyectos:

- Se ha probado a crear una bolsa con un nombre, lo que funciona correctamente.
- Se ha probado a crear una bolsa sin nombre, lo que no funciona y muestra un mensaje de error.
- Se ha probado a introducir como nombre caracteres no alfanuméricos, lo que funciona correctamente.
- Una vez creadas las bolsas, se ha probado a asignar proyectos a esa bolsa y pulsar sobre el botón asignar, lo que funciona correctamente.
- Se ha probado a cambiar proyectos de una bolsa a otra, lo que también funciona.
- Se ha comprobado que un cliente contenga los proyectos asignados cuando se vuelve a cargar la vista.
- Por último se ha probado a desasignar proyectos de una bolsa y se realiza correctamente.

7.3.4 Pruebas de la gestión de movimientos

Para la gestión de movimientos se han realizado todas las pruebas sobre una vista como la siguiente:

Ilustración 72. Pruebas de la gestión de movimientos

Se debe comprobar que:

- Aparezcan todos los clientes.
- Que al seleccionar un cliente, aparezcan todas las bolsas de las que dispone.
- Que al seleccionar una bolsa se carguen todos los movimientos con los que cuenta esa bolsa.
- Que si un movimiento procede de JIRA, se permita reembolsar.
- Que cuando se quiere reembolsar un movimiento, que no se permite reembolsar más horas de las que ese movimiento ha gastado.
- Que si se han rembolsado horas de un movimiento y se desean volver a reembolsar más horas, el total de horas rembolsadas no puede superar a las horas del movimiento.
- Que se permite crear un movimiento introduciendo los valores obligatorios.
- Que no se permite crear un movimiento sin los valores de fecha, título y horas.
- Que el saldo está correctamente calculado
- Que la lista de movimientos está ordenada por fecha.

7.3.5 Pruebas de la visualización de movimientos

Las pruebas realizadas sobre los movimientos que ve un cliente (ilustración 73) solo han requerido que se muestren correctamente las bolsas que contiene un usuario, y que al seleccionar una bolsa, esta contiene los movimientos correspondientes, como ilustra la imagen:

Ilustración 73. Pruebas de la visualización de movimientos

7.3.6 Pruebas de la visualización de estadísticas

Las pruebas realizadas sobre las estadísticas requieren verificar que, si en las bolsas del usuario hay, al menos, un movimiento procedente de JIRA, entonces que muestre las estadísticas (entre ellas, la imagen mostrada a continuación), y si no, que muestre un mensaje que indique al usuario que no tiene movimientos para mostrar estadísticas:

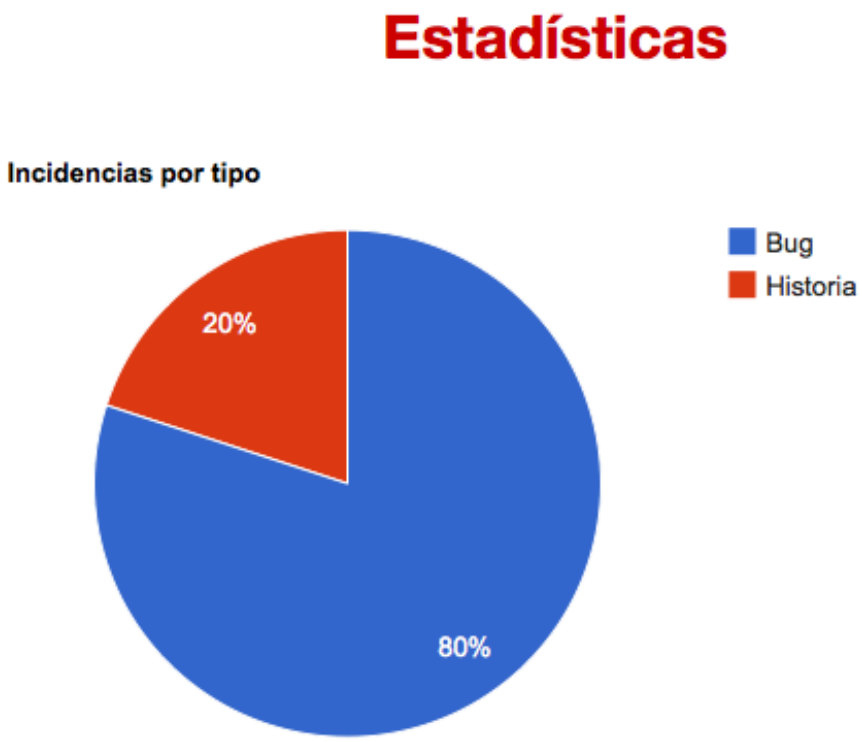

Ilustración 74. Pruebas de la visualización de estadísticas

7.3.7 Pruebas de la visualización de precios

Estas pruebas solo consisten en verificar que cuando se pulsa sobre la opción Precios del menú, se muestra correctamente la lista de precios, como aparece en la siguiente imagen:

Número de horas	Precio	Precio/hora
8 horas	320€	40,00€
40 horas (2% descuento)	1.568,00€	39,20€
80 horas (5% descuento)	3.040,00€	38,00€
160 horas (10% descuento)	5.760,00€	36,00€
320 horas (15% descuento)	10.880,00€	34,00€
640 horas (20% descuento)	20.480,00€	32,00€
1.280 horas (30% descuento)	35.840,00€	28,00€
1.800 horas (40% descuento)	43.200,00€	24,00€

* Estos precios no incluyen IVA

Ilustración 75. Pruebas de la visualización de precios

Sección 8

8 Conclusiones

En esta sección se van a explicar los resultados obtenidos de la realización de este proyecto, comentando las conclusiones alcanzadas y las posibles mejoras a incluir como ampliaciones del proyecto.

8.1 Conclusiones del desarrollo del proyecto

En este apartado vamos a comentar los objetivos conseguidos de los principales puntos a destacar en este desarrollo. Estos puntos son:

- El uso de metodologías ágiles para el desarrollo de un proyecto.
- La gestión del proyecto, con la ayuda de herramientas como JIRA o Confluence.
- Los resultados de utilizar las tecnologías empleadas en este proyecto.
- Las pruebas realizadas a la aplicación resultante.

8.1.1 Uso de metodologías ágiles

Tras concluir este proyecto, podemos afirmar que el uso de las metodologías ágiles ha sido un éxito, destacando especialmente las ventajas que aporta en lo referente a la versatilidad a los cambios o el sencillo ciclo de trabajo en iteraciones que permite encontrar errores y mostrar al cliente los avances rápidamente.

De cara al futuro y en base a nuestra experiencia, esta metodología comenzará a usarse en un mayor número de empresas tecnológicas, y más concretamente en las dedicadas al mundo de la informática, por sus características tan favorecedoras en este tipo de proyectos.

En mi opinión, las metodologías ágiles no son un sustitutivo de las típicas metodologías de desarrollo, como puede ser Métrica v3 [66], sino más bien como una alternativa a valorar, por los beneficios que se obtienen de su uso.

Comparando Métrica v3 con Scrum, podemos decir a favor de estas últimas, que dedican más tiempo al desarrollo del producto que a la redacción de todos los documentos que se requieren, que son más modernas y orientadas al desarrollo software y por tanto, conocen las deficiencias de anteriores modelos y los típicos problemas que se encuentran los proyectos informáticos.

Además, para el uso de Scrum, JIRA dispone de un *plugin* (GreenHopper) que permite gestionar las tareas utilizando esta metodología ágil.

8.1.2 Gestión del proyecto

Para la gestión del proyecto ha sido clave el uso de JIRA como herramienta para gestionar tanto las tareas, como para medir el avance del proyecto y conocer datos sobre las tareas llevadas a cabo, como pueden ser

las fechas de inicio y fin de la tarea, el tiempo dedicado a la misma, que días se ha trabajado en ella, etc.

Esta herramienta está muy difundida en grandes empresas, no solo tecnológicas, sino de todo tipo, como puede ser Facebook, Nike, Ebay, Indra, Everis, Adobe, Twitter, Apache o la NASA. Esto nos hace ver la importancia del uso de este tipo de herramientas en la gestión de proyectos.

Además hay que destacar que esta herramienta está en continuo desarrollo y mejora, por lo que los usuarios disponen de nuevas funcionalidades frecuentemente, sin olvidar que el precio de una licencia para 10 usuarios es de 10 dólares al año, precio asequible tanto para pequeñas empresas, como para particulares que deseen utilizar una herramienta de este estilo.

De la gestión del proyecto también hay que destacar la buena estimación inicial que se realizo y la escasa desviación que finalmente ha resultado, unido a la obtención de casi la totalidad de los beneficios presupuestados inicialmente.

8.1.3 Tecnologías

Las principales tecnologías empleadas en el desarrollo de este proyecto han girado en torno a dos elementos: Groovy y Grails por un lado, y por otro Javascript, que se comentan en sendos apartados a continuación.

Por otro lado, para la implementación de esta aplicación he podido conocer nuevas herramientas, como pueden ser Git o Google Chart Tools. La primera, permite tener el código del proyecto bien estructurado, organizado y disponible en repositorios, para permitir su uso desde múltiples terminales. Por otro lado, Google Chat Tools permite representar gráficas sin necesidad de utilizar herramientas de pago y con la sencillez de utilizar Javascript para ello.

8.1.3.1 Groovy y Grails

Para todos los que no conozcan este lenguaje (Groovy) y este *framework* de desarrollo (Grails), les invito a que lo prueben, especialmente a las personas que ya conozcan Java y lo utilicen asiduamente. Este lenguaje y este *framework* proporcionan la solidez de Java, puesto que se ejecuta sobre la JVM, unido a la sencillez de su sintaxis, junto con una gran librería de *plugins* y extensiones que se pueden utilizar en los proyectos y las continuas mejoras que sufren tanto Groovy como Grails.

Con Groovy podrán seguir utilizando Java como lenguaje de programación o utilizar las facilidades que ofrece este lenguaje, como puede ser el uso de *closures* (segmentos de códigos asignados a una variable que se

ejecutan cuando deseemos), el tipado débil de variables, la legibilidad del código, etcétera.

Con Grails podemos hacer uso del mapeo relacional con la base de datos, lo que nos permite utilizar consultas dinámicas, programar las vistas de una forma más sencilla, mediante las etiquetas que vienen por defecto o creando nuestra propia librería de etiquetas, pasar a la vista los datos desde el controlador de manera sencilla, etcétera.

La unión de este lenguaje y este *framework* ofrece un conjunto de herramientas que encajan a la perfección y que ofrece una característica muy destacada, como es la posibilidad de realizar cambios en tiempo de ejecución, lo que ahorra tiempo durante el desarrollo y permite solucionar errores de manera más rápida.

Antes de comenzar este proyecto tenía experiencia en tecnologías Java y había desarrollado nuevas funcionalidades en proyectos con Groovy y Grails que ya estaban en producción, por lo que este proyecto me ha aportado un gran conocimiento sobre como comenzar un nuevo desarrollo desde cero, y ha mejorado notablemente mis conocimientos sobre el lenguaje y las opciones que aporta el *framework*.

8.1.3.2 Javascript

Este lenguaje de programación continúa siendo uno de los referentes en cuanto a los lenguajes de programación web, indicado, entre otras utilidades, para los retoques visuales y las interfaces de usuario. Gracias a su extensibilidad a través de librerías o *plugins* podemos utilizar múltiples elementos, como menús, pestañas, calendarios, etc.

El uso de este lenguaje continuará estando muy extendido, gracias a su fácil sintaxis, su gran biblioteca de librerías y *plugins* y la cantidad de funciones que se permiten realizar con él.

Gracias a jQuery el uso de Javascript es mucho más sencillo por la cantidad de funciones que implementa esta librería, que es una de las más importantes del lenguaje.

Dentro de la gran cantidad de operaciones que se pueden realizar con jQuery, jQuery-ui nos ofrece una librería de interfaces de usuario con colores personalizables, ampliando así las posibilidades que ofrece jQuery.

Durante el desarrollo de este proyecto he afianzado los conocimientos que ya poseía sobre Javascript y ampliado nuevos conceptos y mecanismos que me han permitido aportar un gran nivel de detalle visual en la aplicación.

8.1.4 Pruebas

La etapa de pruebas es una de las fases más importantes en cualquier proceso de ingeniería, ya que permite conocer si las tareas han sido desempeñadas correctamente o hay algún error y especialmente en la informática, siendo una ciencia tan propensa el error, esta etapa es aún más crítica.

En este proyecto, la etapa de pruebas ha transcurrido a la par que la etapa de desarrollo, lo que nos permite evitar arrastrar errores a lo largo del proyecto, y garantizar que, cuando se termina un *sprint* de Scrum, las tareas que se han terminado funcionan correctamente, tanto en solitario, como interaccionando con las anteriormente realizadas.

Es importante por tanto, que las pruebas realizadas cubran todos los casos, para así asegurar que la funcionalidad está completada y evitar así futuros errores que conlleven un mayor gasto de tiempo y mayores costes para el proyecto.

De todas las pruebas que se deben realizar, hay que hacer especial hincapié en las pruebas unitarias, que permiten verificar el correcto funcionamiento de todos los métodos que contiene nuestra aplicación. Estas pruebas son muy fáciles de implementar y ver donde se encuentra el error del método que están probando.

8.2 Trabajos futuros

Dentro de los trabajos futuros, podemos destacar los trabajos futuros sobre la aplicación que ha sido desarrollada, agregando nuevas funcionalidades.

El desarrollo de las funcionalidades de esta aplicación está limitado por el reducido espacio de tiempo que existe para su conclusión, pero existen nuevas funcionalidades que harían de la aplicación un producto más completo.

Entre esas nuevas funcionalidades que se podrían llevar a cabo están:

- Permitir a los usuarios ver determinadas bolsas de un cliente.
- Habilitar la posibilidad de mover un movimiento de una bolsa a otra, lo que permite mayor flexibilidad a la hora de organizar las bolsas, no solo limitándose a los proyectos.
- Permitir al usuario realizar contrataciones de bolsas desde la propia aplicación a través de:

    - Pasarela de pago bancaria: esta posibilidad es más costosa, puesto que son los propios bancos los que proporcionan las pasarelas de pago y los requisitos para obtener una no son para nada sencillos.
    - Paypal: este mecanismo es mucho más sencillo y fácil de llevar a cabo, pero limita las opciones del usuario a poseer una cuenta Paypal, y no todas las empresas disponen de una, dado que su uso más habitual es en particulares.

- Integrar a la aplicación un sistema de soporte que permita a los usuarios reportar problemas de sus aplicaciones, y mantener un dialogo con los desarrolladores que permita resolver los errores. También permitiría a los usuarios realizar peticiones de cambio o solicitar nuevas funcionalidades. Este sistema de soporte es similar a un foro dedicado para este uso.

8.3 Consideraciones finales

Para acabar las conclusiones, me gustaría destacar varios aspectos aprendidos durante la realización del trabajo de fin de grado de la ingeniería informática.

En primer lugar, destacar la importancia de realizar un proyecto real, orientado a la empresa, creando un producto que tiene un límite de tiempo y unos costes determinados. Esto aporta mayor valor al trabajo realizado.

Además, puesto que se trata de un producto para ser usado, el nivel de detalle está muy cuidado, ofreciendo a los usuarios interfaces sencillas e intuitivas que permitan utilizar la aplicación sin ninguna dificultad.

Otro de los valores a afrontar cuando se trata de una aplicación que va a pasar a la puesta en producción, son las pruebas que verifican que el producto cumple los requisitos y estos están correctamente diseñados e implementados.

En segundo lugar, he aprendido que existen muchas tecnologías para desarrollar aplicaciones informáticas, y que cada una de ella tiene sus propios objetivos, con sus beneficios y sus inconvenientes. Lo importante es no estancarse en conocer un solo lenguaje, sino tener la flexibilidad para querer aprender para conocer varios lenguajes y tecnologías. Esto ofrece al informático mayor capacidad de adaptación a nuevos lenguajes, nuevos proyectos, etc.

En tercer lugar, he descubierto nuevas herramientas que permiten a los desarrolladores emplear menos tiempo para resolver errores o para completar una funcionalidad. Otras herramientas simplifican el desarrollo de ciertas funcionalidades y aportan mayores detalles visuales, ofreciendo un aspecto más conseguido.

En cuarto lugar, he comprendido que no es tan difícil emprender, que lo que hace falta es inculcar a la comunidad docente, a todos los niveles de la educación, la necesidad de potenciar los factores necesarios para los estudiantes se conviertan en emprendedores y creen sus propias empresas. De esta forma se genera una economía de pequeñas y medianas empresas, permitiendo generar puestos de trabajo. Al principio de este proyecto, me encontraba perdido, sin saber muy bien que hacer, pero lo más difícil es encontrar una idea sobre lo qué hacer el proyecto y dar el primer paso. Después llegan una serie de pasos, acertados o no, que te hacen aprender y que mejoran tu experiencia profesional.

En quinto lugar, que el desarrollo del software está sujeto a un continuo cambio y evolución, que requiere estar constantemente leyendo nueva documentación, actualizaciones y conociendo nuevos conceptos, pero también un continuo cambio en los requisitos del sistema, en el diseño o en la forma de implementar los requerimientos.

En sexto lugar, destacar la importancia de las interfaces de usuario. Este aspecto debe tener mayor importancia en el desarrollo de productos, puesto que es la carta de presentación que tienen los usuarios, e influye mucho en la opinión que estos tienen sobre los diferentes desarrollos.

Por último, he continuado adquiriendo experiencia en multitud de campos, como lenguajes de programación, gestores de tareas, herramientas para tener las versiones controladas y otros campos, que me han aportado gran valor de cara a enfrentarme al mundo laboral.

Sección 9

9 Anexo

9.1 Guía para usuarios

Esta guía pretende enseñar a utilizar a los usuarios de la aplicación las diferentes acciones que puede realizar y cómo realizarlas.

En primer lugar, una vez accedemos a la aplicación nos encontraremos con la siguiente pantalla de *login*:

Ilustración 76. Guía de usuarios: *login*

En ella introduciremos el nombre de usuario y la contraseña facilitada por correo electrónico. Si introducimos mal los datos, aparecerá una pantalla de este estilo:

Ilustración 77. Guía de usuarios: *login* incorrecto

Una vez introducimos los datos correctos, nos aparecerá la siguiente pantalla:

Ilustración 78. Guía de usuarios: bolsa

En esta pantalla se nos muestran las bolsas que posee el usuario y el menú con las distintas opciones que posee un usuario. En esta pantalla, una vez seleccionamos la bolsa que queremos ver, aparecerán sus movimientos y el saldo:

Ilustración 79. Guía de usuarios: movimientos de la bolsa

En la opción del menú de las estadísticas, podemos ver una serie de gráficas sobre los movimientos de todas las bolsas de este usuario. En primer lugar se muestran las incidencias por tipo de tarea y en la segunda gráfica podemos ver los movimientos en función del tiempo que se ha empleado en esa tarea.

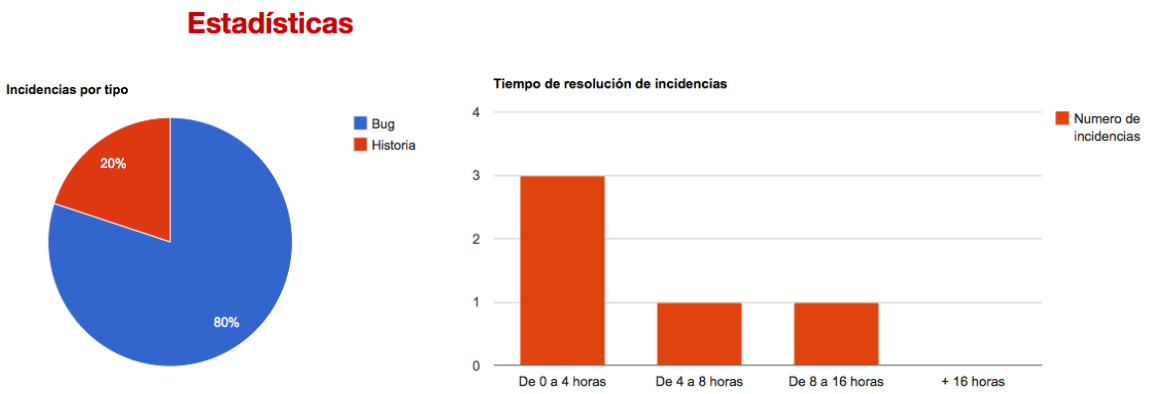

Ilustración 80. Guía de usuarios: estadísticas

Por último, en la opción de precios podemos encontrarnos una pantalla en la que se muestran los precios de las distintas bolsas de horas:

Número de horas	Precio	Precio/hora
8 horas	320€	40,00€
40 horas (2% descuento)	1.568,00€	39,20€
80 horas (5% descuento)	3.040,00€	38,00€
160 horas (10% descuento)	5.760,00€	36,00€
320 horas (15% descuento)	10.880,00€	34,00€
640 horas (20% descuento)	20.480,00€	32,00€
1.280 horas (30% descuento)	35.840,00€	28,00€
1.800 horas (40% descuento)	43.200,00€	24,00€

* Estos precios no incluyen IVA

Ilustración 81. Guía de usuarios: lista de precios

9.2 Guía para administradores

La guía para administradores pretende enseñar a realizar cada una de las acciones que son requeridas para el uso de la aplicación en la parte de administración y resolver aquellas dudas que surjan sobre el manejo de la misma.

Al entrar en la aplicación, aparecerá la ventana de *login*, al igual que en la guía de usuario.

Ilustración 82. Guía para administradores: *login*

Una vez accedemos con los datos correctos, aparecerá la pantalla de integración. En ella podemos configurar los parámetros de JIRA y de Factura Directa, que permiten obtener los datos y unas opciones de configuración de la propia aplicación.

Ilustración 83. Guía para administradores: integración

Para editar los datos de configuración, basta con hacer clic sobre el dato que se desea cambiar y aparece un campo de texto en el que podemos escribir el nuevo valor:

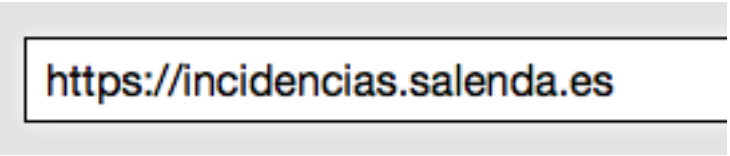

Ilustración 84. Guía para administradores: editar valor integración

Una vez se termina de editar el valor, pulsamos el botón “Enter” y el campo ya esta editado. Si una vez estamos editando un campo decidimos que no queremos hacerlo, basta con pulsar fuera de ese campo.

Si deseamos obtener de nuevo los datos de JIRA o de Factura Directa, solo hace falta pulsar sobre el botón de Refrescar y se obtendrán los datos. Si todo va bien, nos aparecerá un mensaje de esta forma:

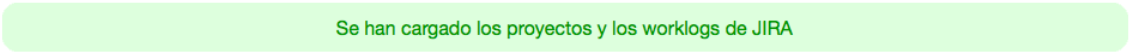

Ilustración 85. . Guía para administradores: integración funciona

Si algo falla, aparecerá este mensaje:

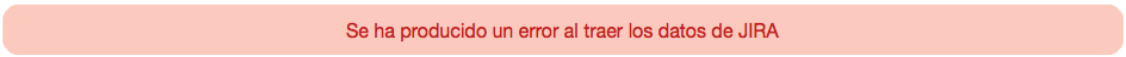

Ilustración 86. . Guía para administradores: error en la integración

En la opción de los usuarios del menú nos encontramos con la siguiente pantalla:

Ilustración 87. Guía para administradores: gestión de usuarios

Desde esta pantalla podemos editar los valores de los usuarios, borrar o agregar nuevos.

Si pulsamos sobre el botón agregar, aparecerá la siguiente pantalla, en la que rellenar la información del usuario. Una vez hecho esto se creará el usuario y se le enviará un correo electrónico con la información de acceso.

Ilustración 88. Guía para administradores: agregar usuario

Si deseamos editar la información, solo tenemos que hacer clic sobre el elemento a editar, al igual que con los parámetros de la vista de integración. Para los roles, puesto que se pueden tener varios, seleccionamos los que queramos y pulsamos sobre el botón “ok”:

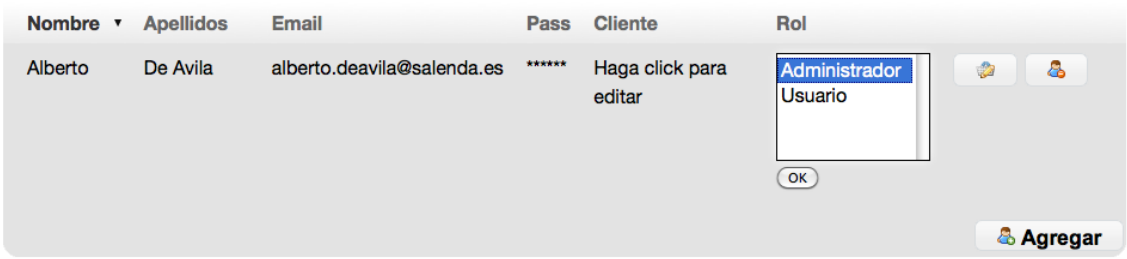

Ilustración 89. Guía para administradores: editar usuario

Si queremos borrar un usuario, pulsamos sobre el botón correspondiente y aparecerá la siguiente imagen:

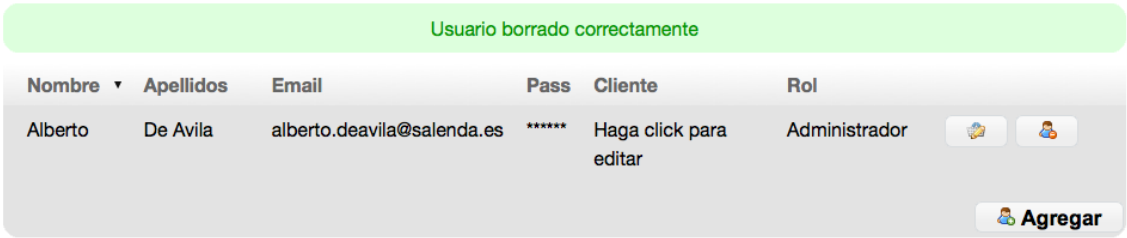

Ilustración 90. Guía para administradores: borrar usuario

En la pantalla de los proyectos, aparecerá la siguiente pantalla:

Ilustración 91. Guía para administradores: proyectos

En la columna de la derecha aparecen los proyectos que no están asignados, y en la de la izquierda, los clientes y sus bolsas. Si deseamos crear una bolsa para un cliente, pulsamos sobre el botón de crear bolsa y nos aparecerá la siguiente pantalla para introducir el nombre de la misma:

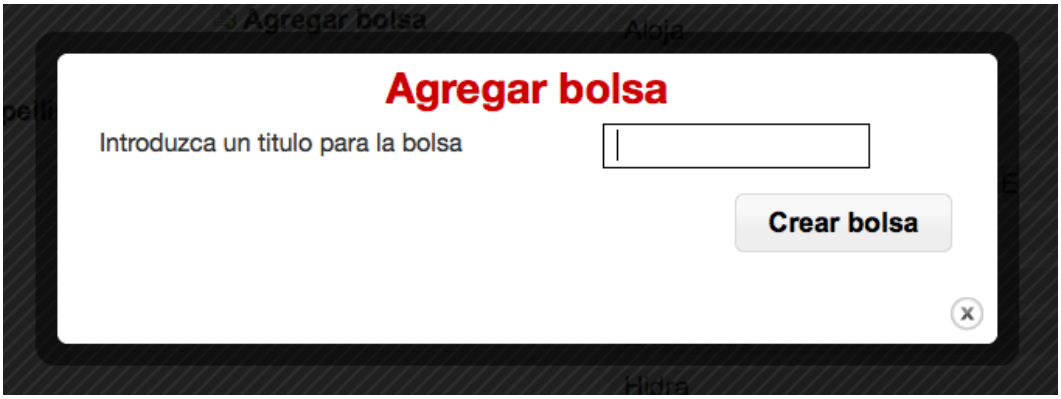

Ilustración 92. Guía para administradores: agregar bolsa

Si introducimos un nombre, al pulsar sobre crear aparecerá el siguiente mensaje:

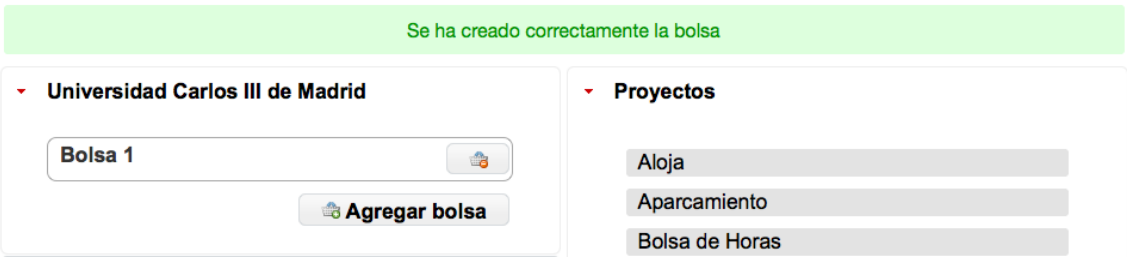

Ilustración 93. Guía para administradores: bolsa agregada

Una vez hecho esto, ya podemos asignar proyectos a bolsas. Para ello, arrastraremos los proyectos a las bolsas donde los queramos asignar. Para facilitar la labor, los campos donde se pueden soltar los proyectos aparecerán en amarillo.

Ilustración 94. Guía para administradores: *drag and drop*

Una vez lo soltamos, aparecerá el proyecto en esa bolsa:

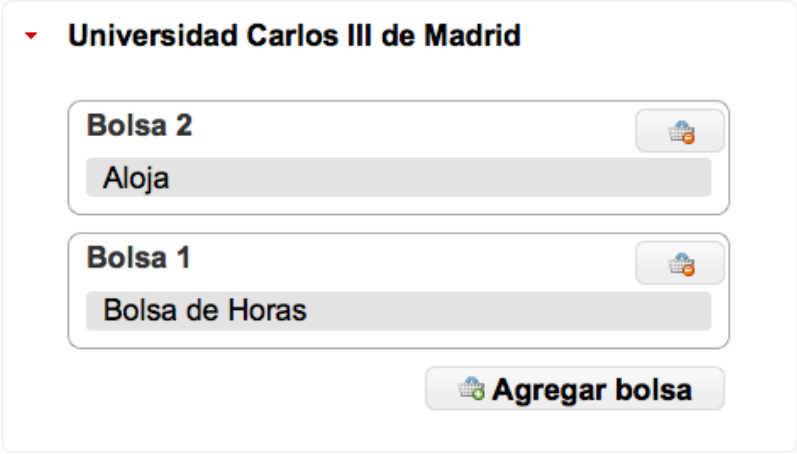

Ilustración 95. Guía para administradores: bolsas con proyectos

Ya solo hace falta pulsar sobre el botón Asignar (abajo a la derecha en la pantalla):

Ilustración 96. Guía para administradores: botón de asignar

Por último, en la opción de Movimientos, podremos ver la siguiente imagen:

Ilustración 97. Guía para administradores: movimientos

En ella se listan todos los clientes de la aplicación. Si queremos gestionar los movimientos de un cliente, pulsamos sobre el elegido y nos aparecerán las bolsas de las que dispone:

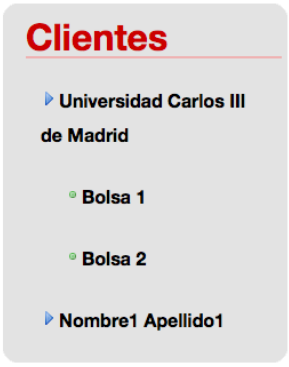

Ilustración 98. Guía para administradores: bolsas del cliente

Seleccionamos la bolsa sobre la que queremos operar, y se cargarán los movimientos que tiene la bolsa. Si la bolsa no tiene movimientos aparecerá la siguiente pantalla:

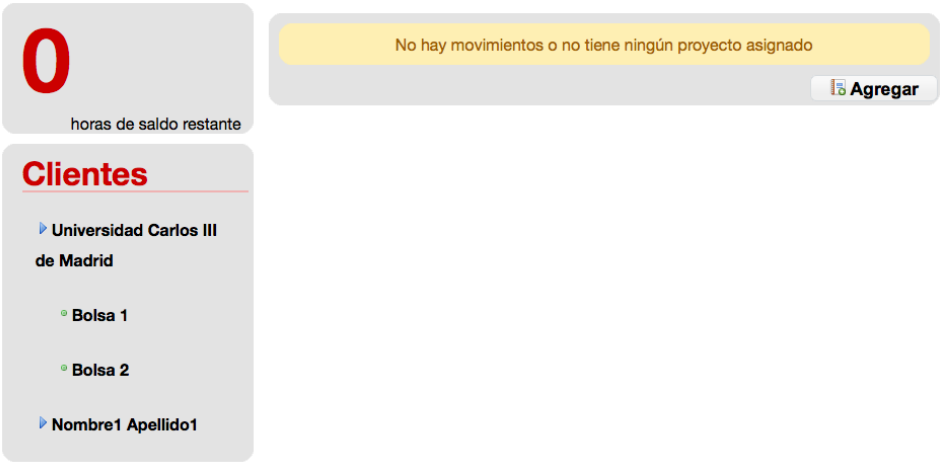

Ilustración 99. Guía para administradores: bolsa sin movimientos

Desde esta nueva ventana podemos crear movimientos de esa bolsa, solo con pulsar el botón Agregar, tras lo cual aparecerá la siguiente pantalla para rellenar los datos necesarios:

Ilustración 100. Guía para administradores: crear movimiento

Los campos obligatorios son la fecha, el título y el número de horas del movimiento.

Cuando se listan los movimientos aparecerán de la siguiente manera:

Ilustración 101. Guía para administradores: listado de movimientos

Si deseamos realizar un reembolso sobre un movimiento, al pulsar sobre el botón correspondiente, se solicitará al administrador el número de horas que desea reembolsar:

Ilustración 102. Guía para administradores: reembolsar movimiento

Una vez introducido, este aparecerá como un movimiento que suma horas:

Ilustración 103. Guía para administradores: movimiento rembolsado

Sección 10

10 Bibliografía y referencias

[1] Salenda (www.salenda.es)
[2] Jira (http://www.atlassian.com/software/jira/)
[3] Confluence (http://www.atlassian.com/software/confluence/)
[4] GreenHopper (http://www.atlassian.com/software/greenhopper/)
[5] H2 (http://www.h2database.com)
[6] Glassfish (http://glassfish.java.net/es/)
[7] Ubuntu (http://www.ubuntu.com/)
[8] SpringSource tool suite (http://www.springsource.com/developer/sts)
[9] Metodologías ágiles (http://www.agile-spain.com/metodos_agiles)
[10] Manifiesto ágil (http://agilemanifesto.org/iso/es/)
[11] Scrum (http://www.scrumalliance.org/)
[12] Jeff, “Lenguaje de Programación”, 16 Octubre 2008, [en línea] http://es.kioskea.net/contents/langages/langages.php3.
[13] Sun Microsystems http://www.oracle.com/es/index.html
[14] ¿Qué es Java? (http://www.java.com/es/download/faq/whatis_java.xml)
[15] PaBLoX, “¿Qué son los lenguajes dinámicos?”, 20 Enero 2010, [en línea] http://www.glatelier.org/2010/01/que-son-los-lenguajes-dinamicos/
[16] Damián Pérez Valdés, ¿Qué es Javascript?, [en línea] http://www.maestrosdelweb.com/editorial/¿que-es-javascript/
[17] Javier Eguíluz Pérez, Introducción a Ajax, [en línea] http://www.librosweb.es/ajax/capitulo1.html
[18] jQuery-ui (http://jqueryui.com/)
[19] Javier J. Gutiérrez, ¿Qué es un framework web?, [en línea] http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf
[20] Frameworks para Groovy, [en línea] http://groovysource.com/open-source/web-frameworks
[21] Grails (http://grails.org/)
[22] Groovy y Grails (http://www.springsource.com/developer/grails)
[23] *Plugin*, [en línea] http://www.masadelante.com/faqs/plug-in
[24] *Plugin* de jQuery para Grails (http://www.grails.org/plugin/jquery)
[25] *Plugin* de jQuery-ui para Grails (http://www.grails.org/plugin/jquery-ui)
[26] *Plugin* Spring Security Core para Grails (http://www.grails-plugins.github.com/grails-spring-security-core/docs/manual/)
[27] *Plugin* Quartz para Grails (http://www.grails.org/plugin/quartz)
[28] *Plugin* Mail para Grails (www.grails.org/plugin/mail)
[29] *Plugin* Settings para Grails (http://www.grails.org/plugin/settings)
[30] *Plugin* Resources para Grails (http://www.grails.org/plugin/resources)
[31] Servidores de aplicaciones, [en línea] http://www.jtech.ua.es/j2ee/2003-2004/abierto-j2ee-2003-2004/sa/sesion1-apuntes.htm
[32] Glassfish (http://glassfish.java.net/es/)
[33] Entornos de desarrollo integrado, [en línea] http://petra.euitio.uniovi.es/~i1667065/HD/documentos/Entornos%20de%20Desarrollo%20Integrado.pdf
[34] JBuilder (http://www.embarcadero.com/products/jbuilder)
[35] Eclipse – I – Historia y Toma de contacto, [en línea] http://www.programacion.com/articulo/eclipse_--_i_--_historia_y_toma_de_contacto_288
[36] SpringSource Tool Suite (http://www.springsource.com/developer/sts)
[37] SpringSource (http://www.springsource.com/)

[38] Microsoft Windows, [en línea]
http://www.maquinariapro.com/sistemas/sistema-operativo-windows.html
[39] Fernando A. Cuenca, Linux, [en línea]
http://www.investigacion.frc.utn.edu.ar/labsis/Publicaciones/QueEsLinux/QueEsLinux.html
[40] Mac OS X (http://www.apple.com/es/macosx/)
[41] vFabric tc Server Developer Edition
(http://www.springsource.com/developer/tcserver)
[42] Tomcat (http://tomcat.apache.org/)
[43] IDE integrados con Grails (http://grails.org/IDE+Integration)
[44] Rafael Navarro Marset, REST, 2006, [en línea]
http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf
[45] API REST de Twitter (https://dev.twitter.com/docs/api)
[46] Atlassian (http://www.atlassian.com/)
[47] Eduardo Mayor, JIRA Plugins: JQL en JIRA, 22 Noviembre 2011, [en línea]
http://www.novagenia.com/blog/2011/11/22/jira-plugins-jql-en-jira/
[48] Dulce introducción al Kanban!!, Andoni Arroyo, 12 Septiembre 2009, [en línea] http://geeks.ms/blogs/aarroyo/archive/2009/08/12/kanban.aspx
[49] Microsoft Office (http://office.microsoft.com/es-es/)
[50] Firebug (https://addons.mozilla.org/es-es/firefox/addon/firebug/)
[51] Mozilla Firefox (http://www.mozilla.org/es-ES/firefox/new/)
[52] Firebug Lite
(https://chrome.google.com/webstore/detail/bmagokdooijbeehmkpknfglimnifench/details)
[53] Google Chrome (http://www.google.es/chrome)
[54] HTML (http://www.desarrolloweb.com/articulos/que-es-html.html)
[55] GIT (http://www.git-scm.com)
[56] Casos de uso, [en línea]
http://www.dcc.uchile.cl/~psalinas/uml/casosuso.html
[57] MVC, XEROX PARC, 1978 -1979, [en línea]
http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html
[58] Cocoa (https://developer.apple.com/technologies/mac/cocoa.html)
[59] Ruby on Rails (http://rubyonrails.org/)
[60] Struts (http://struts.apache.org/)
[61] JSF (http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html)
[62] Django (https://www.djangoproject.com/)
[63] Python (http://www.python.org/)
[64] ASP .NET MVC (http://www.asp.net/mvc)
[65] UML (http://www.uml.org/)
[66] Métrica v3
(http://administracionelectronica.gob.es/?_nfpb=true&_pageLabel=P60085901274201580632&langPae=es)